MÓDULO 8 – Conceitos Avançados de Programação

Objetivos do módulo:

- Entender as especificidades da programação em ambiente gráfico.
- Constatar as diferenças entre a programação procedimental e a programação por eventos.
- Conhecer a interface de programação do sistema operativo.
- Tomar conhecimento dos problemas associados à interface com o utilizador no desenvolvimento de aplicações para ambientes gráficos.

Definição

1. Vantagens de um sistema operativo gráfico
   Um sistema operativo gráfico apresenta vantagens óbvias para o utilizador. Para o programador as vantagens não sendo tão óbvias são muitas, a começar pela quantidade de código fornecido ao programador pelo sistema operativo (SO), todos os elementos gráficos existentes no SO estão disponíveis para o programador reutilizar. Num ambiente não gráfico o programador tem de criar os elementos da interface.

2. Conceito de janela
   Num SO de ambiente gráfico a janela é o elemento básico de cada programa. Um programa tem de ter pelo menos uma janela, ainda que a possa esconder. A janela é redesenhada sempre que necessário pelo SO não sendo necessário promover refrescamentos mesmo que a janela fica parcialmente sobreposta.

3. Conceitos acerca da interface com o utilizador
   O Windows apresenta um vasto conjunto de funcionalidades, que permite a qualquer programador, através de uma linguagem de programação, criar a interface com o utilizador. Esta caraterística do SO é designada por GDI (Graphics Device Interface), sendo responsável por tarefas básicas como desenhar linhas, curvas e tipos de letra, providenciando um conjunto de APIs (Application Programming Interface) para a execução dessas operações.

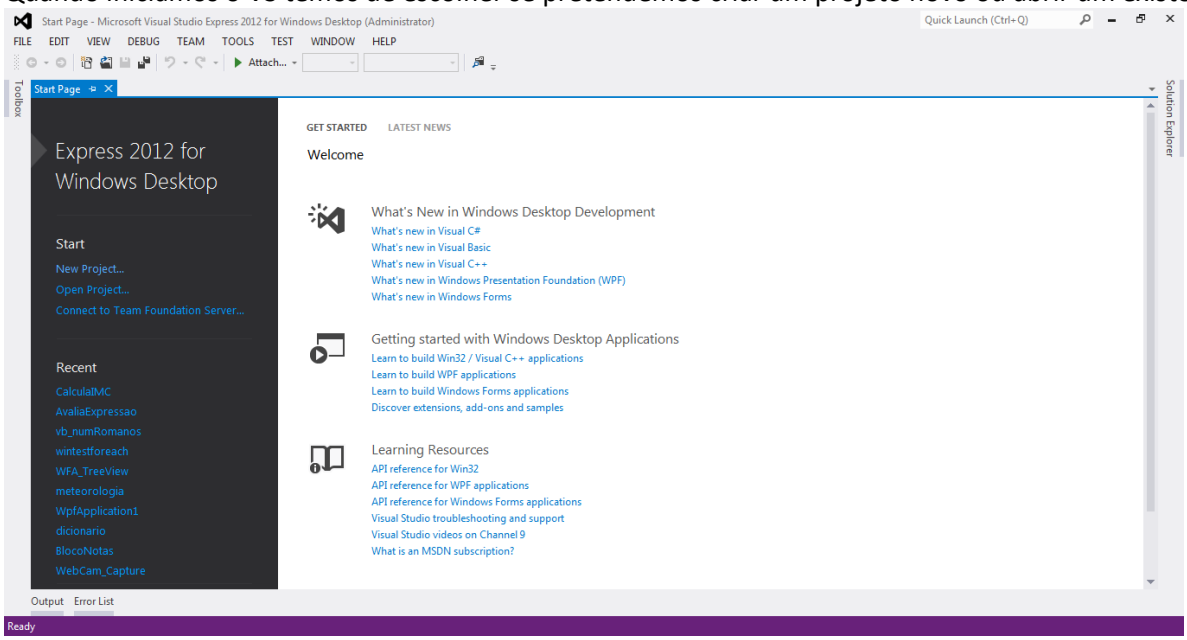
   A parte visível de qualquer aplicação (formulários, caixas de diálogo, botões, menus, barras de ferramentas, imagens, etc) é implementada pela interface da própria linguagem visual a que o programador recorre através do seu IDE (Integrated Development Environment) bastando clicar e arrastar os elementos para a sua posição.

   A linguagem de programação utilizada é responsável por aceder ao GDI e criar os objetos que o programador solicitar, ou seja, ao arrastar de um elemento para a interface corresponde a instruções geradas automaticamente na linguagem de programação escolhida.

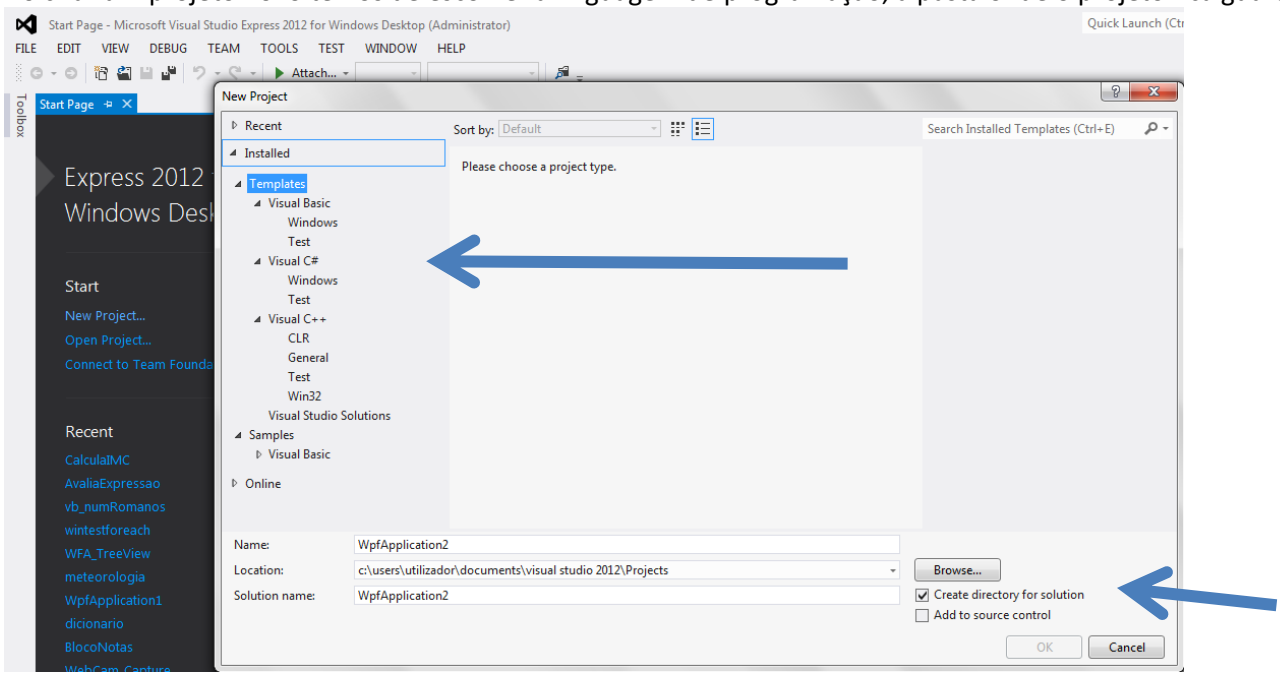
   Ao pensar o programa devemos ter em conta a disponibilidade da interface mesmo quando estão a ser executadas outras operações de cálculos intensos, ou seja, a interface com o utilizador não deve ficar "pendurada" à espera que o programa termine de fazer outra coisa, para isso existem as threads que permitem lançar ações demoradas em segundo plano. Este tipo de tarefas obriga que o programador pondere quais os elementos gráficos disponíveis para o utilizador bem como o seu estado em diferentes momentos.

   O IDE a utilizar neste e nos próximos módulos é o Visual Studio Community. Trata-se de uma ferramenta gratuita disponível para download no site da Microsoft.

Quando iniciamos o VS temos de escolher se pretendemos criar um projeto novo ou abrir um existente:

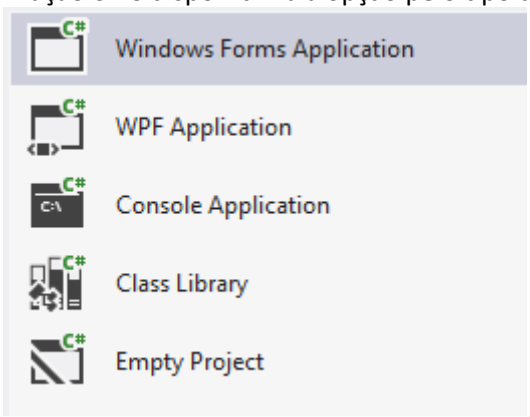


Ao criar um projeto novo temos de escolher a linguagem de programação, a pasta onde o projeto fica guardado.

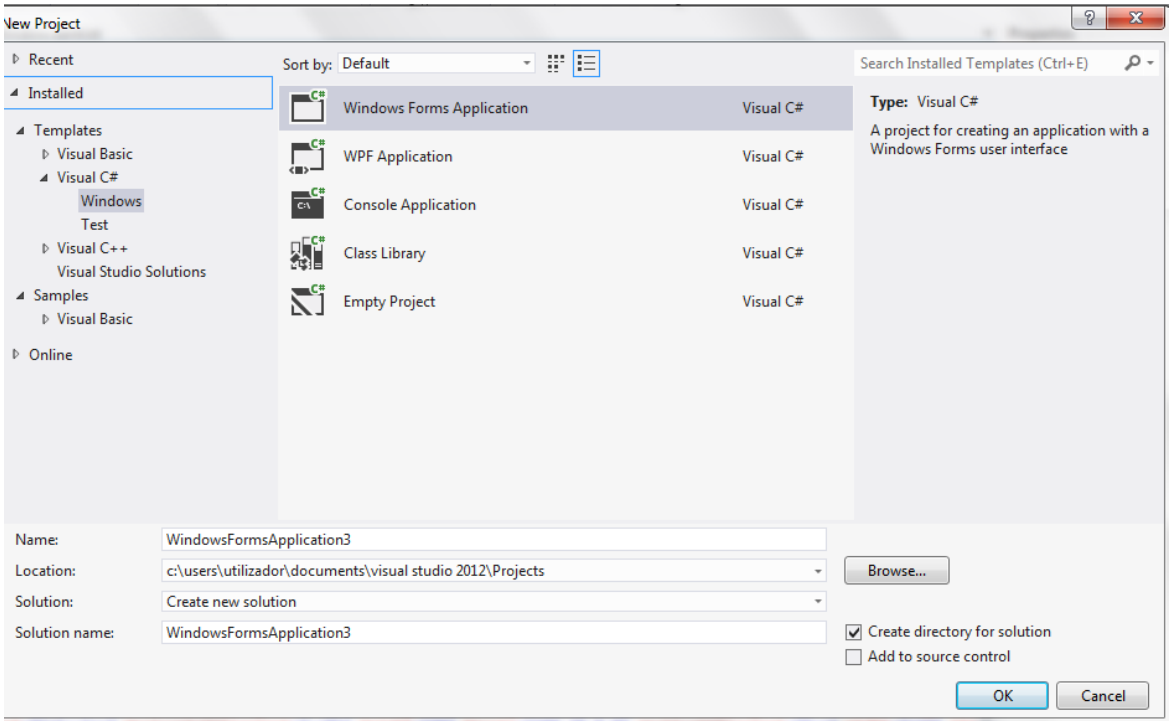


Nas versões anteriores existiam versões independentes para cada uma das linguagens de programação.

Após escolher a linguagem de programação o VS disponibiliza a opção pelo tipo de aplicação a criar. As opções são:



Windows Forms Application – uma aplicação para Windows que utiliza formulários tradicionais
WPF Application – uma aplicação para Windows com recurso à nova interface
Console Application – uma aplicação para Windows em modo consola
Class Library – Uma DLL
Empty Project – projeto vazio

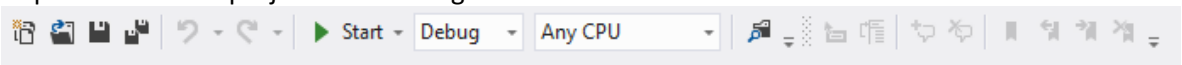


Ao criar um projeto novo o VS cria uma solução para esse projeto. Um projeto corresponde a um programa, uma solução pode incluir vários projetos.

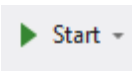
Ao criar um programa novo para ambiente gráfico o processo passa por definir os seguintes passos:

1. Definir a interface gráfica (GUI).
2. Definir as propriedades dos objetos utilizados na GUI.
3. Codificar os eventos a que estes objetos vão reagir.

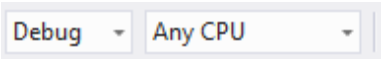
Depois de criado o projeto temos a seguinte barra de ferramentas:



Os primeiros botões permitem criar um ficheiro novo, abrir um ficheiro existente e guardar o atual ou todos os ficheiros.

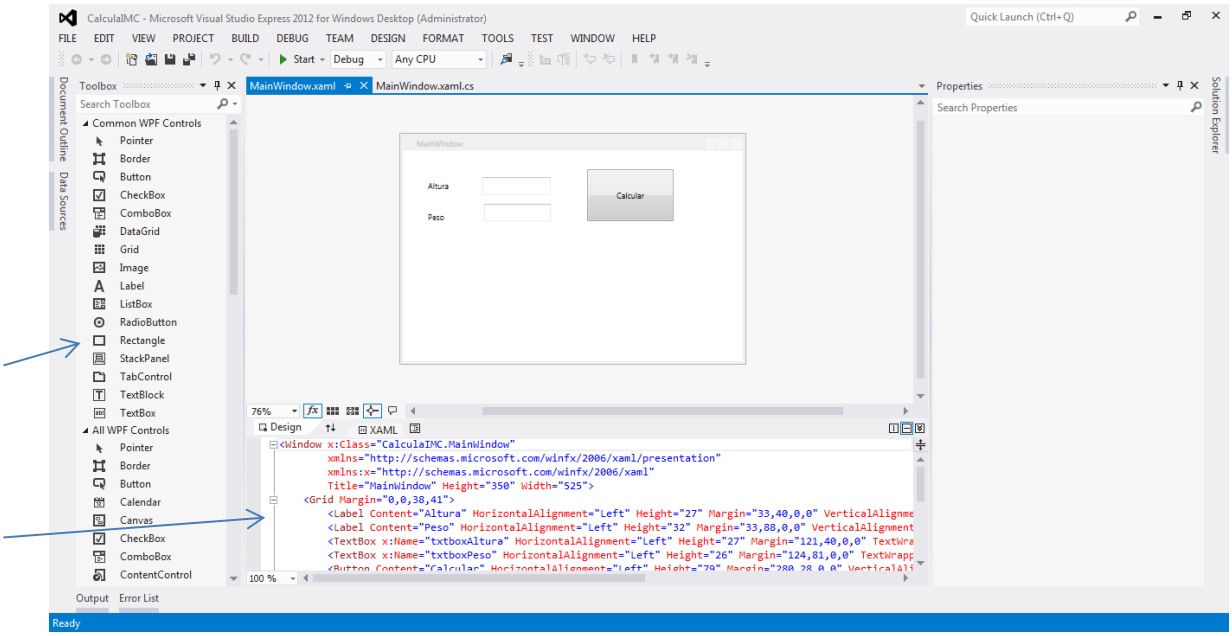


Executa o programa



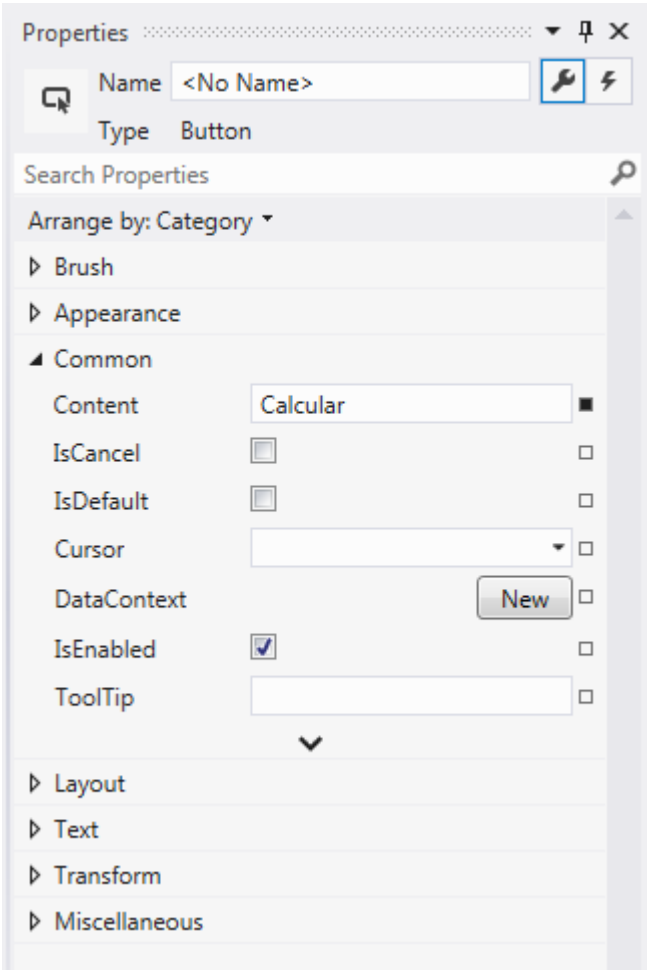
Modo de execução do programa e o CPU (32 ou 64 bits)

Quando estamos a criar a interface gráfico o aspeto é o seguinte:

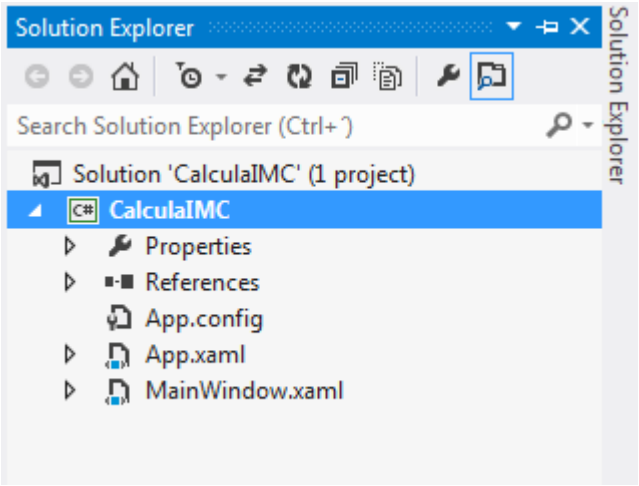


Do lado esquerdo temos uma barra de elementos que podemos incluir na janela do programa.
Na parte de baixo é apresentado o código XAML necessário para gerar a interface.
Do lado direito existem duas barras importantes, a primeira apresenta as propriedades do objeto selecionado, a segunda apresenta os ficheiros que fazem parte do projeto e da solução.

Propriedades de um botão num programa WPF



Solution Explorer



4. **Programação por eventos e "queues"**
   Os programas em ambiente Windows não interagem diretamente com o *hardware*. O Windows, através dos drivers, faz a gestão dos dispositivos físicos e envia às aplicações mensagens com os eventos e estado dos mesmos, por exemplo, quando o utilizador prime uma tecla o Windows envia ao programa ativo o evento *KeyPress* e passa-lhe as informações necessárias para que o programa possa responder a esse evento.

   Os programas não têm de responder a todos os eventos, o Windows define uma fila (*queue*) de mensagens para cada programa. Os programas têm um ciclo de leitura das mensagens a partir do qual é dada uma resposta ou não aos eventos. Nesse ciclo as mensagens lidas são retiradas da fila, mesmo quando o programa não reage ao evento.

   Quando o programa bloqueia ou executa uma tarefa de processamento intenso pode acontecer que não retire as mensagens da fila durante algum tempo (5 segundos no caso do SO Android) e isso é entendido pelo SO como um programa que não responde, dando ao utilizador a oportunidade de forçar o seu encerramento.

   Os programas desenvolvidos com o VS não necessitam de explicitamente implementar o ciclo de leitura/retirada das mensagens do Windows, esse ciclo é criado pelo VS bastando ao programador definir a quais eventos o seu programa vai reagir e o que faz.

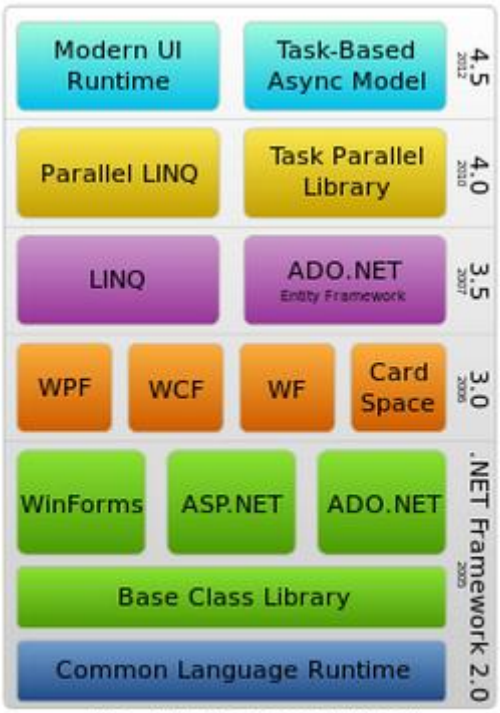
5. Conceitos relativos à interface de desenvolvimento de aplicações (API) do sistema operativo
   Os sistemas operativos modernos (Windows, Linux, Android, MacOS) implementam um conjunto de interfaces (API) que permitem ao programador criar os seus próprios programas. As linguagens de programação definem funções para o programador utilizar escondendo as chamadas a essas interfaces e tornando a programação mais simples, muitos casos bastando clicar e arrastar.

   As APIs do Windows estão definidas em ficheiros DLL que acompanham o SO e no seu próprio núcleo (Kernel) no caso de funções fundamentais como seja gestão de memória.

   A Microsoft ao longo dos anos tem tentado tornar os seus SOs mais seguros e estáveis, por vezes os erros apresentados pelo Windows são provocados por programas mal escritos que tentam executar operações inválidas ou que não gerem corretamente erros em tempo de execução.

   Com esse objetivo em mente foi criada uma estrutura comum de programação (*framework*) que pode ser utilizada por diferentes linguagens de programação permitindo partilhar código entre si bem como tornar esse código mais seguro. Essa estrutura também permite que o mesmo código seja compatível entre diferentes

versões do SO bastando para isso atualizar a *framework*. Ao longo dos anos essa estrutura evoluiu estando atualmente na versão 4.5.



The .NET Framework Stack

Overview of .NET Framework release history

Generation	Version number	Release date	Development tool	Distributed with
1.0	1.0.3705.0	2002-02-13	Visual Studio .NET	N/A
1.1	1.1.4322.573	2003-04-24	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005	Windows Server 2003 R2
3.0	3.0.4506.30	2006-11-06	Expression Blend	Windows Vista, Windows Server 2008
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	2010-04-12	Visual Studio 2010	N/A
4.5	4.5.50709.17929	2012-08-15	Visual Studio 2012	Windows 8, Windows Server 2012

É dentro desta framework que os programas desenvolvidos com o VS são executados.
Existe um projeto para implementar esta framework em ambiente Linux denominado Mono. Este projeto é muito interessante pois permite que um executável possa ser utilizado no Windows bem como no Linux sem necessidade de recompilar ou alterar o código. O IDE que permite programar para este ambiente é gratuito e pode ser utilizado no Windows, no MacOS ou no Linux. O seu nome é MonoDevelop, está atualmente na versão 4.0, e já permite também criar aplicações para Android.

6. O modelo de memória

Dentro do Windows cada programa tem o seu espaço de endereçamento virtual de memória, quer isto dizer que para cada programa existe um espaço reservado de memória, é esta funcionalidade que permite que quando um programa bloqueia os restantes não sejam afetados.
Um processo ou programa de 32bits pode utilizar até 4 gigabytes de memória, em ambiente 64bits o espaço vai até 8 terabytes!
Cada programa só pode aceder ao seu próprio espaço, se um programa tentar violar esta regra é bloqueado. Podem existir, no entanto, espaços de memória partilhados entre processos.
Como o espaço de endereçamento é virtual os endereços utilizados pelo programa não correspondem ao endereço físico da memória, o SO faz a tradução do endereçamento interno do programa com o espaço físico.
O espaço de endereçamento da memória é divido em duas partições, uma parte para o programa outra para o SO.

7. Introdução à programação em C#
   a. Estrutura de um programa em C# modo consola

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

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Análise do código

O programa em C# começa por indicar as namespaces que utiliza. Uma namespace inclui código que pode ser reutilizado por outros programas.

Todos os programas têm a sua própria namespace, isto permite que o código possa ser utilizado noutro projeto.

Dentro da namespace tem de existir uma class. Uma class é uma estrutura que inclui variáveis e código.

Nessa class tem de existir a função Main, tal como nos programas em C, esta é a função onde a execução começa.

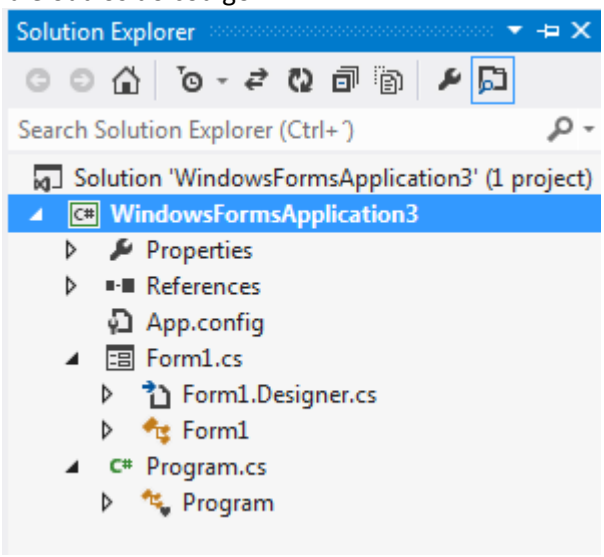
Para interagir com a consola temos algumas instruções associadas a esta:

Console.WriteLine() – escreve na janela da consola

Console.Read() – devolve o conteúdo introduzido pelo teclado

   b. Estrutura de um programa em C# Windows Forms

Uma aplicação Windows Forms ou WPF é constituída por vários ficheiros sendo alguns da interface do programa e outros do código.



Alguns destes ficheiros são criados e manipulados pelo VS e o programador menos experiente não deve editar o seu conteúdo.

A seguir apresento o código do ficheiro Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication3
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Análise do código

Como podemos ver este ficheiro é gerado pelo VS e apenas define o estilo visual da aplicação e executa o Form1, o formulário inicial da aplicação.

A seguir apresento o código associado ao Form1.cs

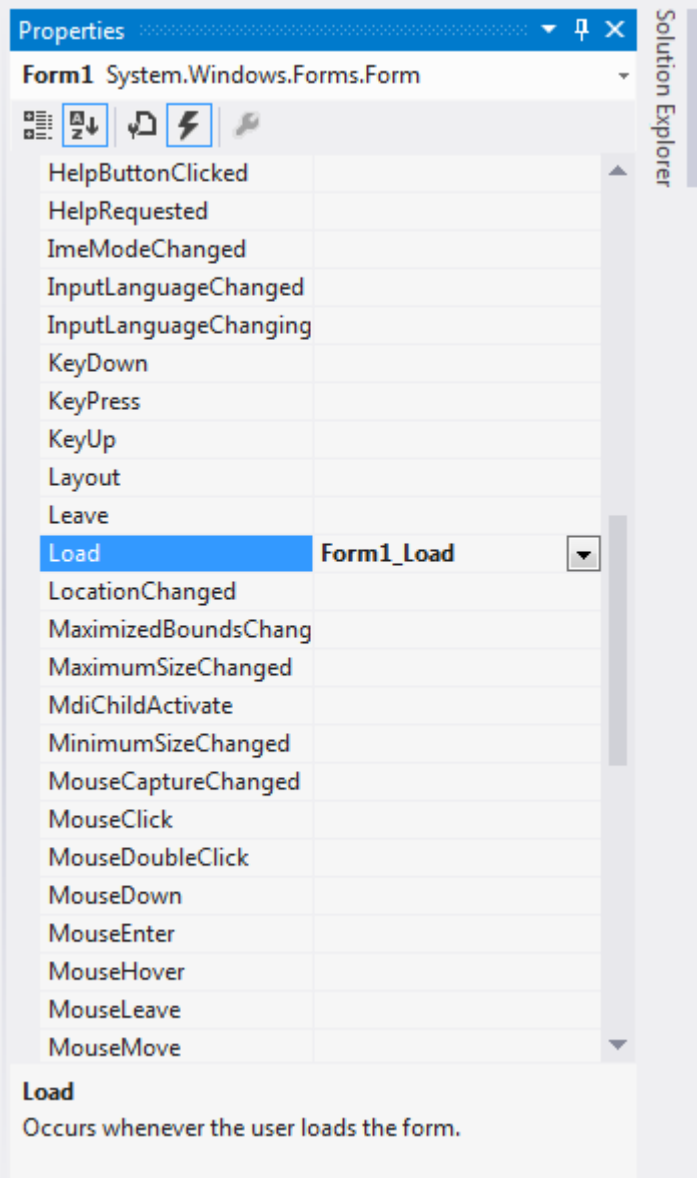
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Análise do código

Este código apenas inicializa os componentes do formulário e apresenta uma função Form1_Load que não tem nenhum código. Esta função corresponde a um evento, mais precisamente ao evento load do formulário, ou seja, o seu código é executado quando o formulário é carregado na memória ainda antes de ser apresentado ao utilizador.

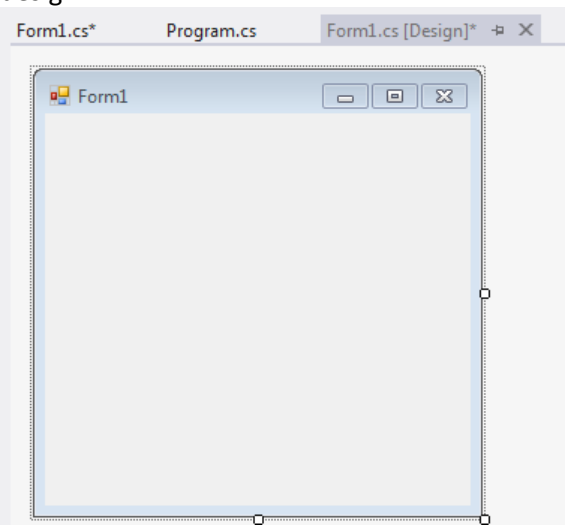
Para definirmos os eventos dos nossos objetos podemos proceder de duas maneiras: 1. Fazemos duplo clique sobre o objeto, por exemplo um duplo clique na barra de título do formulário cria o evento load. 2. Na barra das propriedades existe um botão que faz aparecer a lista de eventos que pode ser

programados para o objeto selecionado, assim basta procurar o evento pretendido e fazer duplo clique sobre ele.

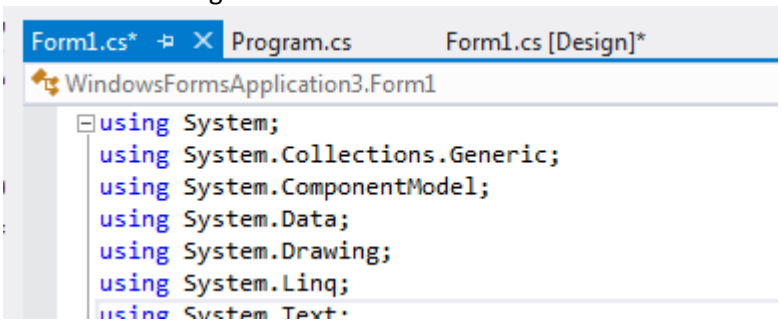


Neste modo podemos definir o nome da função e ainda podemos consultar uma pequena ajuda que explica quando o evento ocorre.

Quando trabalhamos com um formulário existem dois modos em que este pode ser visualizado. O modo design:



E o modo de código dos eventos



O primeiro modo utiliza-se para definir a interface o segundo para codificar o comportamento da aplicação.

c. Tipos de dados e constantes

A seguinte tabela apresenta os tipos de dados mais comuns do C#

Tipo de dados	Armazena…
Byte, USHORT, UINT e ULONG	Números inteiros positivos
SBYTE, SHORT, INT e LONG	Número inteiros (positivos e negativos)
FLOAT, DOUBLE e DECIMAL	Números inteiros e reais (positivos e negativos)
CHAR e STRING	Carateres e cadeias de carateres
BOOL	Valores lógicos

Alguns exemplos, não esquecer que o C# é case sensitive:

```
byte valor;

int numero;

float taxa;

double x;

decimal preco;

char letra;

string nome;

bool verdadeiro;
```

Podemos declarar e atribuir valores às variáveis na mesma linha. Ao contrário de C existe o tipo de dados string que permite guardar vários carateres.
Os comentários são iguais aos utilizados em C.
Também podemos declarar várias variáveis do mesmo tipo separando-as com vírgulas.
Muito importante:
As variáveis declaradas dentro das funções só existem dentro das funções, são locais. Se pretendermos criar variáveis globais devemos defini-las no topo da classe logo abaixo da linha da sua definição class…

Vetores e matrizes
Para definirmos um vetor utilizamos a seguinte sintaxe:
tipo[] nome = new tipo[tamanho];

Exemplos:

```
string[] alunos = new string[7];

alunos[0] = "Joaquim";
alunos[1] = "Daniel";

alunos[7] = "Maria";
```

A última linha dá erro! Porquê?

Para definirmos uma matriz com duas dimensões utilizamos a seguinte sintaxe:
tipo[,] nome = new tipo[tamanho1,tamanho2];
Uma matriz pode ter mais do que duas dimensões.

Exemplos:

```
string[,] dados = new string[7,2];

alunos[0,0] = "Joaquim";
alunos[0,1] = "Santa Comba Dão";

alunos[1,0] = "Daniel";
alunos[1,1] = 3500;
```

Será que existe aqui algum erro? Qual?

Os vetores em C# são mais "inteligentes" pois têm algumas propriedades e funções que podemos utilizar:

d. Operadores aritméticos

Os operadores aritméticos do C# são basicamente os mesmo do C:

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira

e. Operadores relacionais

De igual modo os operadores relacionais são semelhantes aos do C:

Operador	Operação
==	Igual
!=	Diferente
<	Menor que
>	Maior que
<=	Menor ou igual que
>=	Maior ou igual que

f. Operadores lógicos

Mais uma vez, em C e em C# tudo muito parecido:

Operador	Operação
&&	E lógico
||	Ou lógico
!	Negação

g. Funções de controlo de fluxo

Estruturas de decisão

Tal como em C temos o if e o switch para decidir a condição de execução de instruções:

```
if(condição)
{
    [bloco de instruções]
}
```

As {} chavetas permitem definir blocos de código, tanto em condições como em ciclos e funções.

```
If(condição)
{
}
else
{
}

switch(expressão)
{
case valor1:
            [bloco de instruções]
            break;
case valor2:
            [bloco de instruções]
            break;
default:
    [bloco de instruções]
}
```

Estruturas de repetição

Os ciclos em C# funcionam da mesma maneira que em C, existe, no entanto, um novo:

```
while(condição)
{
    [bloco de instruções]
}

do
{
    [bloco de instruções]
} while(condição);

for(inicio;condição;atualização)
{
    [bloco de instruções]
}
```

O ciclo foreach permite criar uma estrutura de repetição a aplicar a cada elemento de um conjunto de dados, por exemplo um vetor:

```
foreach(elemento in estrutura)
{
    [bloco de instruções]
}
```

Exemplo:

```
int[] vetor = new int[]{10,9,8,7,6,5};
int soma = 0;

foreach (int valor in vetor)
{
     soma += valor;
}

MessageBox.Show(soma.ToString());
```

h. Funções

Apesar do C# ser uma linguagem de programação orientada aos objetos também permite criar funções. Estas têm a mesma estrutura base das funções em C. Assim:

tipo nome_função(parâmetros)
{
    [bloco de instruções]
    return valor;
}

Exemplo:

```
void aviso(string texto)
{
     MessageBox.Show(texto);
}
```

Bibliografia

Programa da disciplina
http://en.wikipedia.org/wiki/.NET_Framework
http://msdn.microsoft.com/en-us/library/aa366525(VS.85).aspx
C#5.0 com Visual Studio 2012, Curso Completo, FCA