Departamento de Engenharia Informática

Instituto Superior de Engenharia do Porto

Instituto Politécnico do Porto

Introdução ao Desenvolvimento de Aplicações Baseadas em Componentes

Usando as plataformas .NET e Java

Paulo Sousa

—♦—

Outubro de 2005

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto (ISEP/IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 PORTO
Portugal
Tel. +351 228 340 500
Fax +351 228 325 219

Criado em Outubro, 2003
Última modificação em 27 Outubro, 2005 **(v 2.0.1)**
Email: psousa@dei.isep.ipp.pt
Blog: http://spaces.msn.com/members/pgsousa/

URL: http://www.dei.isep.ipp.pt/~psousa/aulas/ADAV/guiao_componentes.pdf

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

Este documento pretende introduzir os aspectos relacionados com o desenvolvimento de aplicações segundo o paradigma dos componentes.

Os exemplos de código e guiões de trabalho recorrem à plataforma .NET, ao Visual Studio e à linguagem C#, mas o importante são os conceitos apresentados e não a tecnologia específica. Esses conceitos podem ser aplicados praticamente em qualquer plataforma e linguagem de programação.

Todos os exemplos deste guião podem ser feitos usando o Visual Studio, o .Net Framework SDK da Microsoft ou o Projecto Mono (implementação de código aberto - *open source* – multiplataforma da especificação CLI e compatível com o CLR 1.1)

No Anexo 1 encontra informação sobre como utilizar o SDK do .net para desenvolvimento de aplicações sem necessitar de usar o Visual Studio (o SDK está disponível de forma gratuita para download). No Anexo 2 encontra uma breve introdução à linguagem C# e alguns links para sítios com informação adicional.

No Anexo 3 encontra informação sobre como utilizar a plataforma Java para implementar os conceitos apresentados neste documento, bem como o código fonte dos guiões de trabalho do componente de aritmética (capítulos 4, 5 e 6) e de carregamento dinâmico de componentes (capítulo 7).

2 A Plataforma .net

A plataforma .net é um ambiente de execução, CLR (*Common Language Runtime*) (Figura 1) e uma plataforma de desenvolvimento, *base class library* (Figura 2), i.e., conjunto de classes base sobre a qual se desenvolve

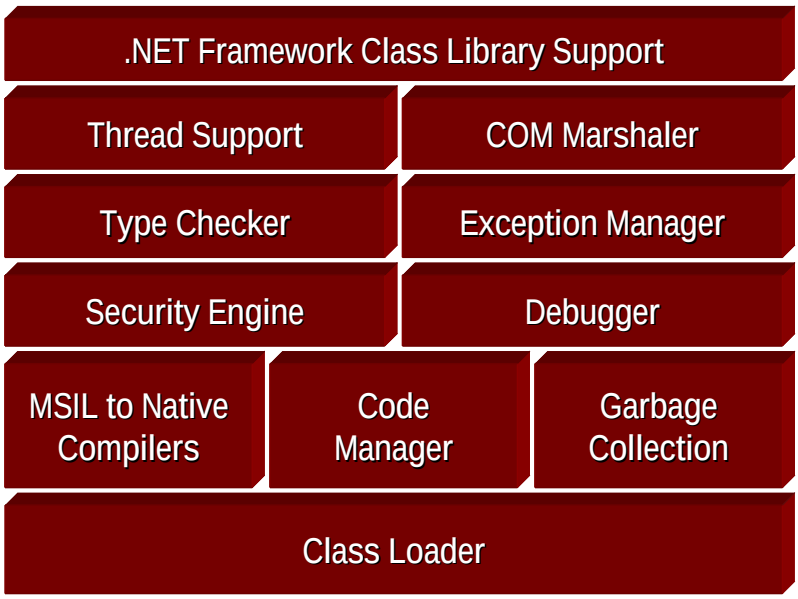

Figura 1 – CLR

As classes da biblioteca de classes base do .net estão organizadas em *namespaces*[1] (Figura 2). Um *namespace* é um "espaço de nomes", ou seja, um contexto de identificação de tipos que permite organizar o código de forma estruturada (hierárquica) e agrupar logicamente tipos (ex., classes) relacionadas. Por exemplo, todas as classes base para comunicações via rede estão dentro do *namespace* `System.Net` ou em *namespaces* dentro desse em caso de especialização (ex., `System.Net.Sockets`). Os *namespaces* não estão relacionados com herança nem com os *assemblies* (é possível ter classes de um mesmo *namespace* em *assemblies* diferentes).

Os *namespaces* são uma técnica de organizar o código e evitar conflito de nomes. Por exemplo, ao desenvolver aplicações web ou Windows é possível criar páginas com controlos do tipo *text box*, no entanto a classe que representa uma *text box* é diferente para cada um dos tipos de aplicações. As classes podem ter o mesmo nome, ex. `TextBox`, desde que estejam em *namespaces* diferentes, ex., `System.Web.UI` e `System.Windows.Forms`.

[1] Para ver um mapa com toda a hierarquia de *namespaces* da plataforma .net consultar o URL http://msdn.microsoft.com/library/en-us/cpref/html/cpref_start.asp.

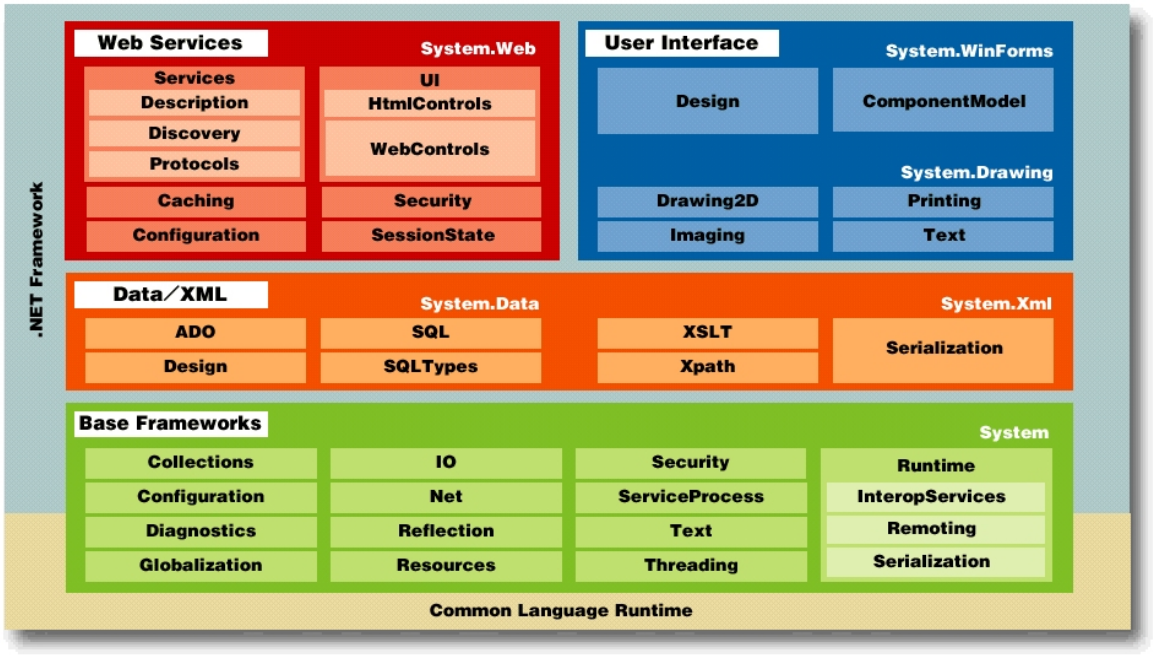

Figura 2 – namespaces existentes na plataforma .net

Para se referirem no código a uma classe devem utilizar o nome completo da classe (incluindo o *namespace*) . Por exemplo:

```
System.String x = “olá mundo”;
System.Random r = new System.Random();
```

No entanto, para simplificar a codificação, é possível indicar ao compilador que se deseja utilizar classes de um determinado *namespace*, usando a directiva `using` (`import` em visual basic.net), e dessa forma é possível referir essas classes sem o nome completo. Por exemplo:

```
using System;
...
String x = “olá mundo”;
Random r = new Random();
```

Desta forma a escrita do código fica mais simplificada.

Um outro conceito importante na plataforma .net é o conceito de ***assembly***. Um *assembly* é um bloco de distribuição binária da aplicação (pode ser um executável ou uma DLL), que contém uma colecção de tipos e de recursos (ex., imagens JPEG), é “versionavel”, e ao qual podem ser afectadas permissões de segurança no contexto .net.

A figura seguinte representa o processo de execução de um programa na plataforma .net.

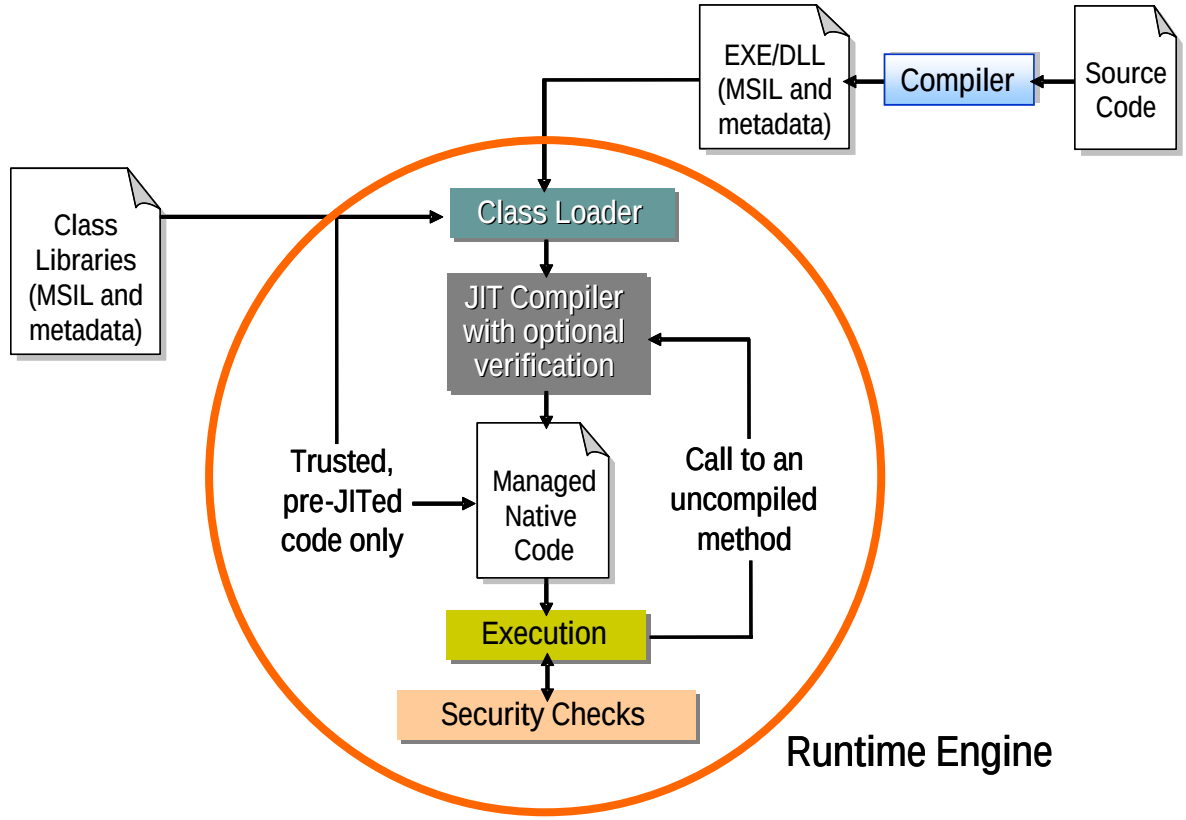

Figura 3 – processo de compilação e execução em .NET

Os tipos de dados dividem-se em duas categorias:

- *Value type*: contém directamente os dados e **não** pode ser `null`
- *Reference type*: contém referência para objecto e pode ser `null`

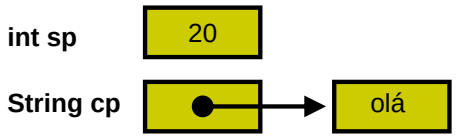

Figura 4 - exemplo de um value type e um reference type

Como a maior parte das variáveis dos programas .net são tipos de referência e não de valor, é necessário ter alguns cuidados na atribuição e comparação de variáveis. Veja-se por exemplo o seguinte extracto de código C# :

```
Pessoa p1 = new Pessoa(“antónio”);
Pessoa p2;
p2 = p1;
p2.Nome = “joão”;
System.Console.WriteLine(“p1.Nome : {0}”, p1.Nome);
```

Como as variáveis de classe são referências, a atribuição **não copia** os objectos. Ambas as variáveis referenciam o mesmo objecto, logo a última linha vai escrever no ecrã:

```
p1.Nome : joão
```

e não:

```
p1.Nome : António
```

Na figura seguinte podemos ver graficamente aquilo que acabou de ser explicado:

```
Pessoa p1 = new Pessoa(“António”);
Pessoa p2;
p2 = p1;
p2.Nome = “joão”;
```

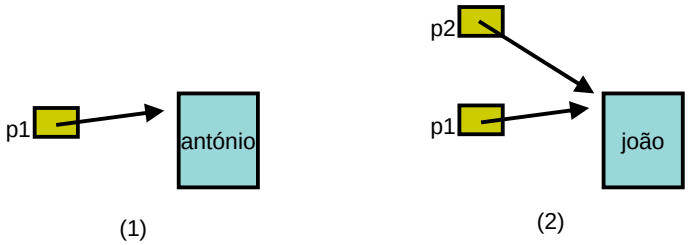

Figura 5 – atribuição de referências

Um outro cuidado adicional tem a ver com os testes de igualdade entre variáveis. Como as variáveis são referências, o operador de igualdade[2] é na realidade um teste de **identidade**, ou seja, verifica se as duas variáveis referenciam o mesmo objecto.

```
Pessoa p1 = new Pessoa(“António”);
Pessoa p2 = p1;
Pessoa p3 = new Pessoa(“António”);
Pessoa p4 = new Pessoa(“luisa”);

System.Console.WriteLine(“p1 == p2 : {0}”, p1 == p2);
System.Console.WriteLine(“p1 == p3 : {0}”, p1 == p3);
System.Console.WriteLine(“p1 == p4 : {0}”, p1 == p4);
```

[2] Em C#, além do operador de igualdade (i.e., ==) é possível comparar dois objectos usando o método `Equals()` definido na classe `Object`. Para mais informação ver http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemObjectClassEqualsTopic1.asp?frame=true. De notar, para os programadores Java, que em Java o operador == efectua um teste de identidade enquanto o método `equals()` efectua um teste de igualdade. Em C#, tanto o operador == como o método `Equals()` têm o mesmo comportamento semântico.

O extracto de código anterior produziria o seguinte conteúdo:

```
p1 == p2 : true
p1 == p3 : false
p1 == p4 : false
```

No entanto, em determinadas situações, o que o programador deseja é um teste de **igualdade de valor** e não de identidade de objectos. Nessa situação teremos que definir explicitamente um método para comparação por valor ou redefinir a semântica do método `Equals()` da nossa classe.

Supondo que a classe pessoa tem dois atributos (nome e data de nascimento) a comparação entre dois objectos `Pessoa` seria implementada da seguinte forma:

```
public override bool Equals(object o)
{
  if (o is Pessoa)
  {
    if (this.nome == o.nome && this.dtNasc == o.dtNasc)
      return true;
    else
      return false;
  }
  else
    return false;
}
```

Como foi dito anteriormente, em C#, o operador == e o método `Equals()` têm o mesmo comportamento semântico. Assim sendo, ao redefinir um deles devemos também redefinir o outro. Neste caso, deveria acrescentar-se à classe `Pessoa` o seguinte operador:

```
public static bool operator==(Pessoa p1, Pessoa p2)
{
  if (p1 != null && p2 != null)
  {
    if (p1.nome == p2.nome && p1.dtNasc == p2.dtNasc)
      return true;
    else
      return false;
  }
  else
    return (p1 == null && p2 == null ? true : false);
}
```

De notar que segundo as regras para redefinição de operadores ao redefinir o operador == também deve ser redefinido o operador `!=` (para mais informação ver http://msdn.microsoft.com/library//en-us/csref/html/vcwlkoperatoroverloadingtutorial.asp).

3 Desenvolvimento utilizando o Visual Studio .Net 2003

3.1 IDE

O ambiente de desenvolvimento do Visual Studio .Net 2003 (Figura 6) permite geração de código nativo windows e .net, no qual é possível criar soluções[3] multi-projecto (em que cada projecto pode ser numa linguagem de programação distinta). Permite a criação de diferentes tipos de projecto (ex., aplicações de linha de comando, aplicações windows e aplicações web).

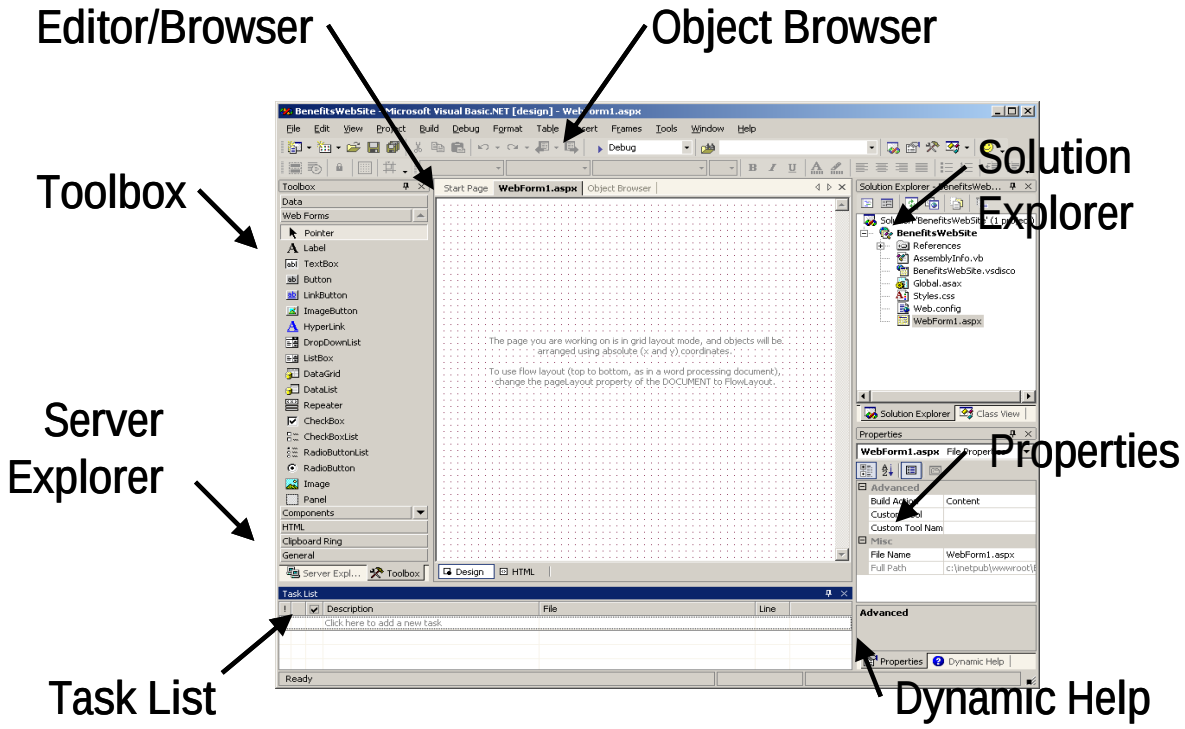

Figura 6 – Visual Studio .net

3.2 "Olá Mundo" em Consola

Criar um novo projecto do tipo "Console Application" em C#

[3] No visual studio, uma solução (*solution*) funciona como um espaço de trabalho no qual podem existir vários projectos.

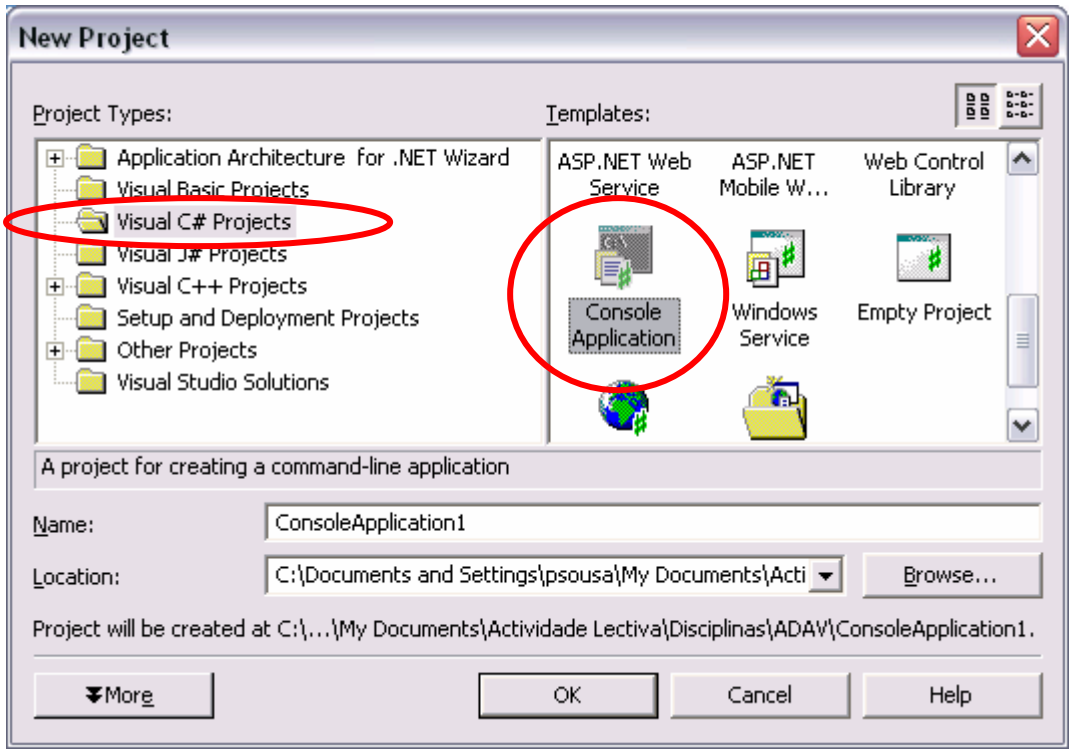

Figura 7 – criar um projecto tipo consola em C#

O Visual Studio vai gerar uma solução com um projecto contendo vários ficheiros. No ficheiro “class1.cs” existe já uma classe definida com um método `Main()`.

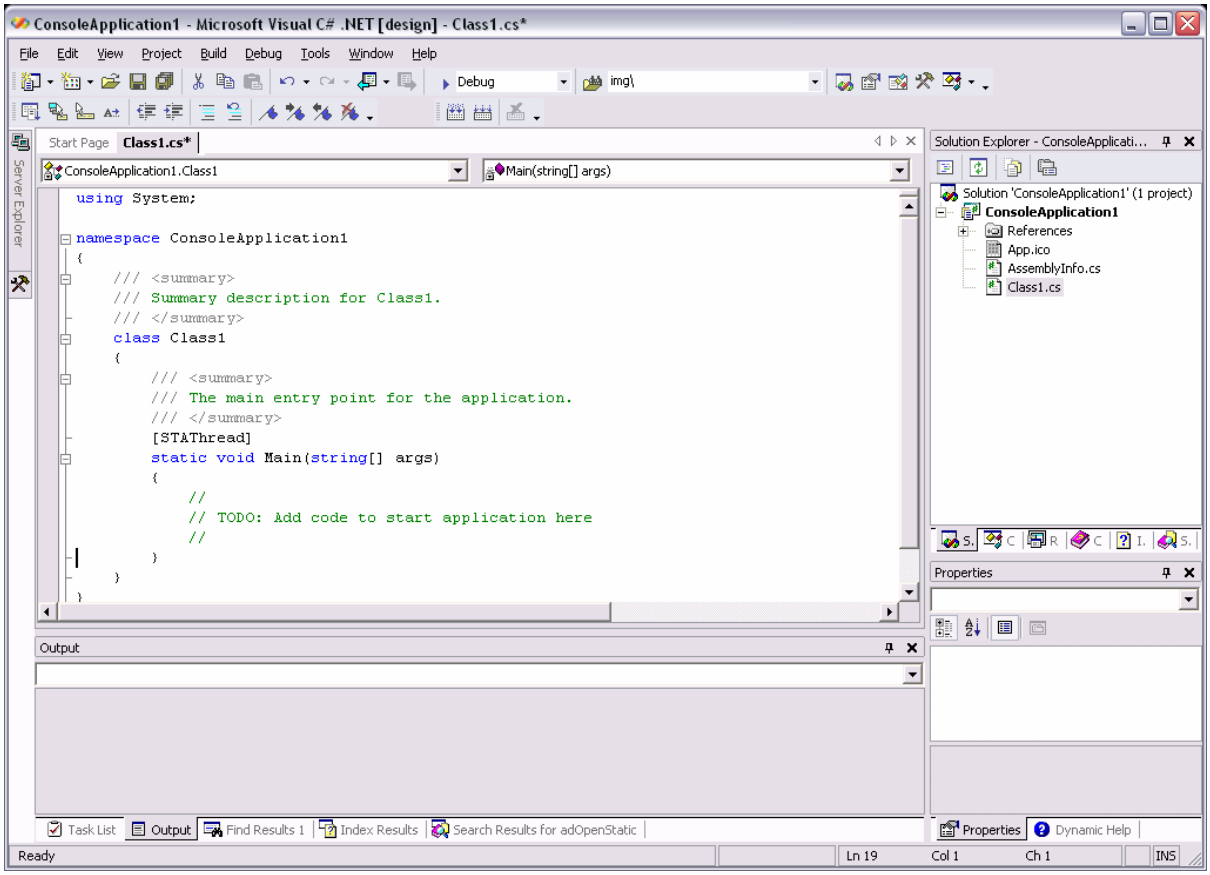

Figura 8 – código gerado automaticamente para projectos consola em C#

Colocar na implementação do método `Main()` a seguinte linha de código.

```
System.Console.WriteLine("Olá mundo!");
```

Compilar e testar.

Caso se queira criar novas classes no projecto deve-se “clicar” com o botão do lado direito do rato em cima do projecto, escolher a opção Add → Add Class e criar uma classe (Figura 9) ou escolher a opção File → Add New Item e escolher o *template* “Class” da categoria “Local Project Itens” (Figura 9).

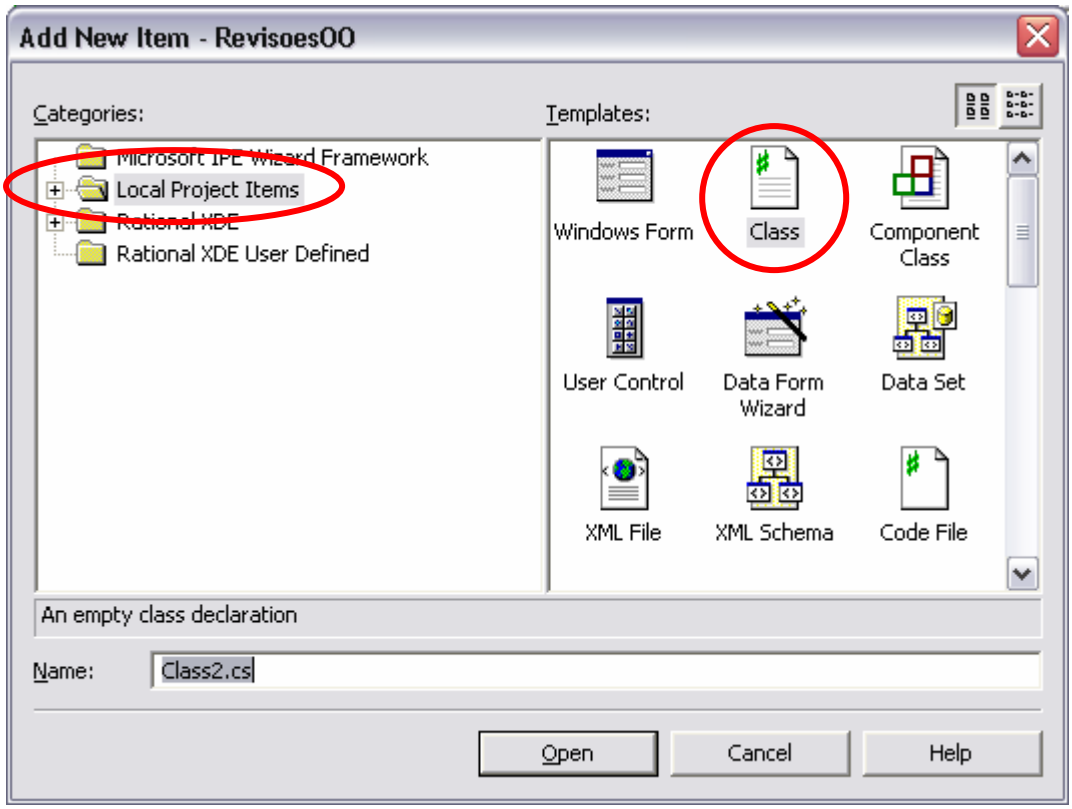

Figura 9 – adicionar uma nova classe a um projecto

No Visual Studio, o *solution explorer* permite visualizar a estrutura do projecto em termos de ficheiros (Figura 10) enquanto o *class explorer* permite visualizar a estrutura do projecto em termos de *namespaces* e classes (Figura 11).

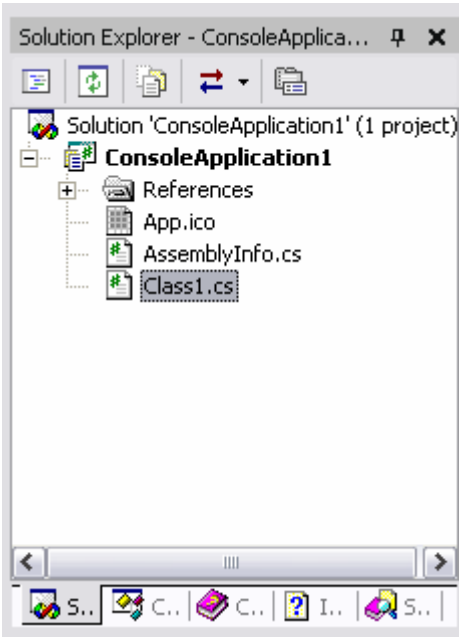

Figura 10 – solution explorer

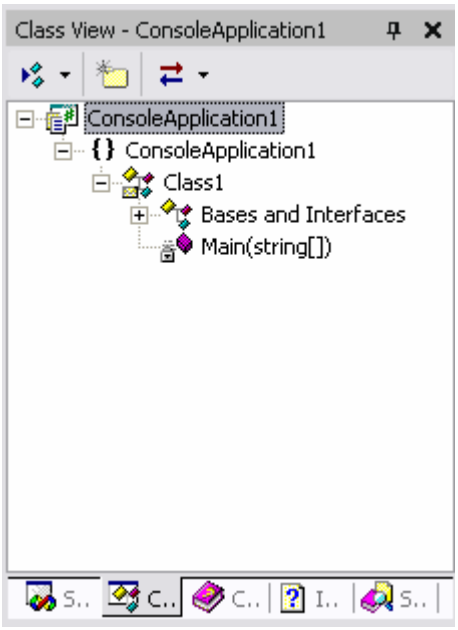

Figura 11 – class explorer

As classes criadas no Visual Studio pertencem sempre a um *namespace*. Por omissão o Visual Studio cria um *namespace* com o mesmo nome do projecto (que é também por omissão o nome do *assembly* a gerar), no entanto é possível modificar o *namespace* para novas classes nas propriedades do projecto (Figura 12).

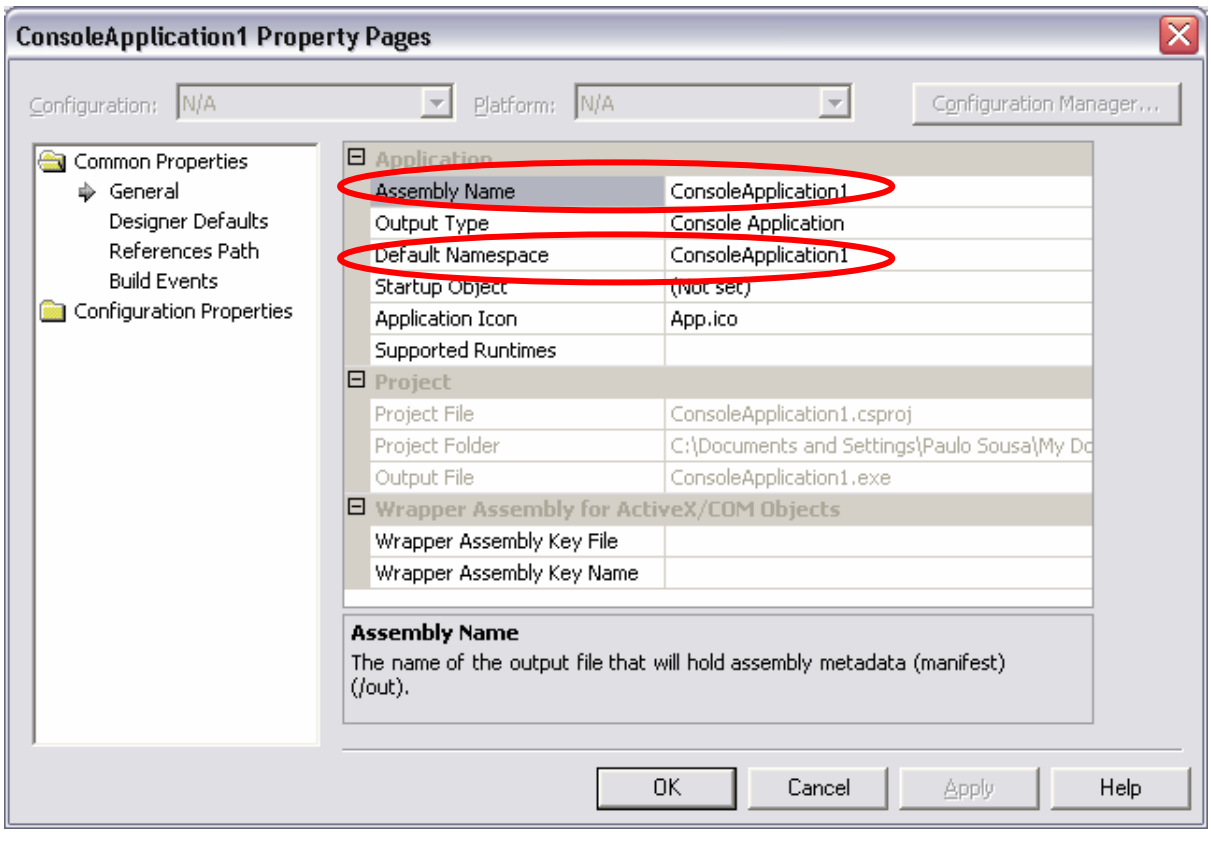

Figura 12 – propriedades de um projecto no Visual Studio (1)

Nos laboratórios do DEI, dará jeito criarem os projectos na vossa área de trabalho na rede, mas dirigir o resultado de compilação para um directório temporário na máquina local. Para isso podem alterar nas propriedades do projecto o “output path” (Figura 13).

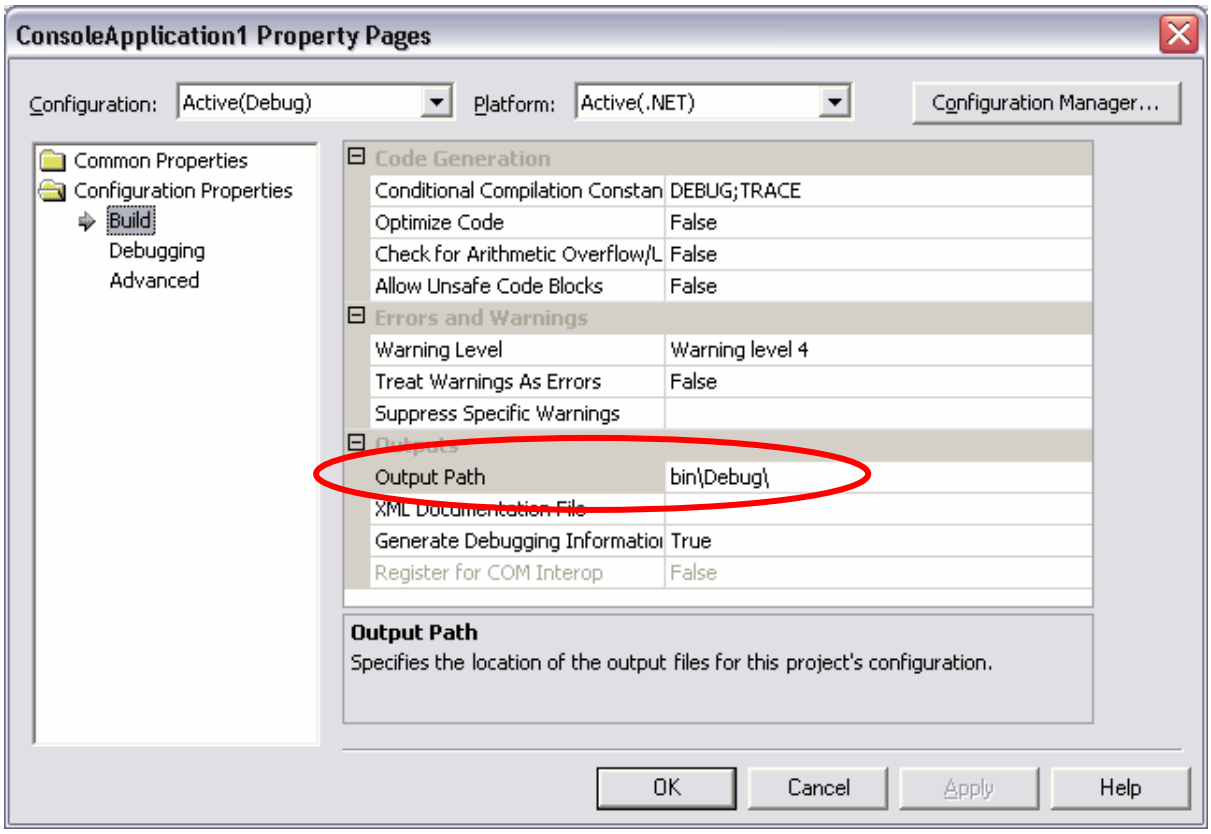

Figura 13 – propriedades de um projecto no Visual Studio (2)

3.3 Exercícios

1 Escreva em C# o código que define uma classe de objectos `Pessoa` que representa pessoas. Considere atributos como ‘nome’, ‘data de nascimento’, BI e NIF. Implemente os métodos que julgar necessários não esquecendo que os atributos são privados a cada objecto. Escreva ainda um pequeno programa que permita criar objectos desta classe.

2 Tal como no exercício anterior, escreva em C# o código que define uma classe de objectos `Endereco` que representa endereços. Especifique os atributos necessários e implemente os métodos necessários.

3 Usando as duas classes de objectos dos dois exercícios anteriores escreva também em C# o código que define uma classe de objectos `Contacto` que representa contactos de pessoas. Não se esqueça de incluir atributos como o telefone. Reflicta sobre quais os atributos desta classe, bem como, quais os

parâmetros aconselháveis para cada um dos seus métodos à luz do conceito OO de encapsulamento e do facto da linguagem usar tipos de referência e não tipos de valor. Deve também escrever um pequeno programa que permita criar objectos desta classe.

4 Implemente em C# uma classe de objectos que permita armazenar um conjunto de referências a objectos `Contacto`. Internamente estes objectos são armazenados numa colecção do tipo `System.Collections.ArrayList` (procure no *help* do Visual Studio informação sobre como utilizar esta classe). Esta classe deve ter, além dos construtores necessários, os seguintes métodos:

`Add` – adiciona um contacto à lista

`AddAt` – adiciona um contacto numa dada ordem

`At` – devolve o contacto numa dada posição

`Find` – devolve o 1º contacto que encontra que possua a chave de pesquisa desejava (decida que campo utilziar como chave de pesquisa)..

Reflicta sobre quais os parâmetros aconselháveis para cada um destes métodos à luz do conceito OO de encapsulamento. Escreva um pequeno programa que crie pelo menos um objecto desta classe, insira contactos e procure por um dado contacto.

4 Guião de trabalho: Componente para operações aritméticas

4.1 introdução

Antes de avançar para o desenvolvimento de uma aplicação baseada em componentes em .net convém relembrar o conceito de componente e verificar como é que tal pode ser mapeado na tecnologia .net.

Segundo alguns autores, um componente é:

- "A software package which offers **service** through **interfaces**" [4]

[4] Peter Herzum and Oliver Sims, "Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise", John Wiley & Sons, Incorporated, 1999.

- "A coherent package of software artifacts that can be independently developed and **delivered as a unit** and that can be **composed**, unchanged, with other components to build something larger" [5]
- "A component is a **unit of composition with contractually specified interfaces** and explicit context dependencies only. A software component can be **deployed independently** and is subject to **composition** by third parties." [6]

De acordo com estas definições, **um componente é uma unidade de distribuição em formato binário de código, que fornece serviços através de um contrato bem definido** (i.e., uma interface). Por sua vez, **uma interface é um conjunto coerente de métodos**. Por conjunto coerente entenda-se métodos relacionados com um mesmo serviço e/ou trabalhando com entidades de um mesmo domínio. Um componente é utilizado por terceiros aos quais se dá o nome de "cliente", **um cliente é uma aplicação ou outro componente que faz uso dos serviços de um componente**.

Em .net para responder a estes requisitos temos (Tabela 1):

Tabela 1 – mapeamento entre requisitos para componentes e a tecnologia .net

Requisito	Mapeamento .net
Unidade de distribuição binária	*Class library*
Fornecimento de serviços	Classes
Contrato explícito	Interfaces
Reutilização	Referência à *class library*
Composição por terceiros	Composição e especialização (herança) de classes

Para uma breve introdução a estes conceitos aconselha-se a consulta do glossário disponível na Microsoft Developer Network Library[7] em http://msdn.microsoft.com/library/en-us/netstart/html/cpconGlossary.asp.

[5] D.F. D'Souza and A.C. Wills, "Objects, Components, And Frameworks with UML – The Catalysis Approach" Addison-Wesley, 1998.

[6] C. Szyperski, "Component Software: Beyond Object-Oriented Programming" Addison-Wesley, 1998.

[7] MSDN : http://www.msdn.microsoft.com/library

4.2 Solução

No Visual Studio, uma solução (*solution*) funciona como um espaço de trabalho no qual podem existir vários projectos de diferentes tipos (executáveis, bibliotecas de classes, etc.) para um mesmo problema. Vamos então criar uma solução em branco (i.e., sem projectos), usando a opção de menu File → New → Blank solution .

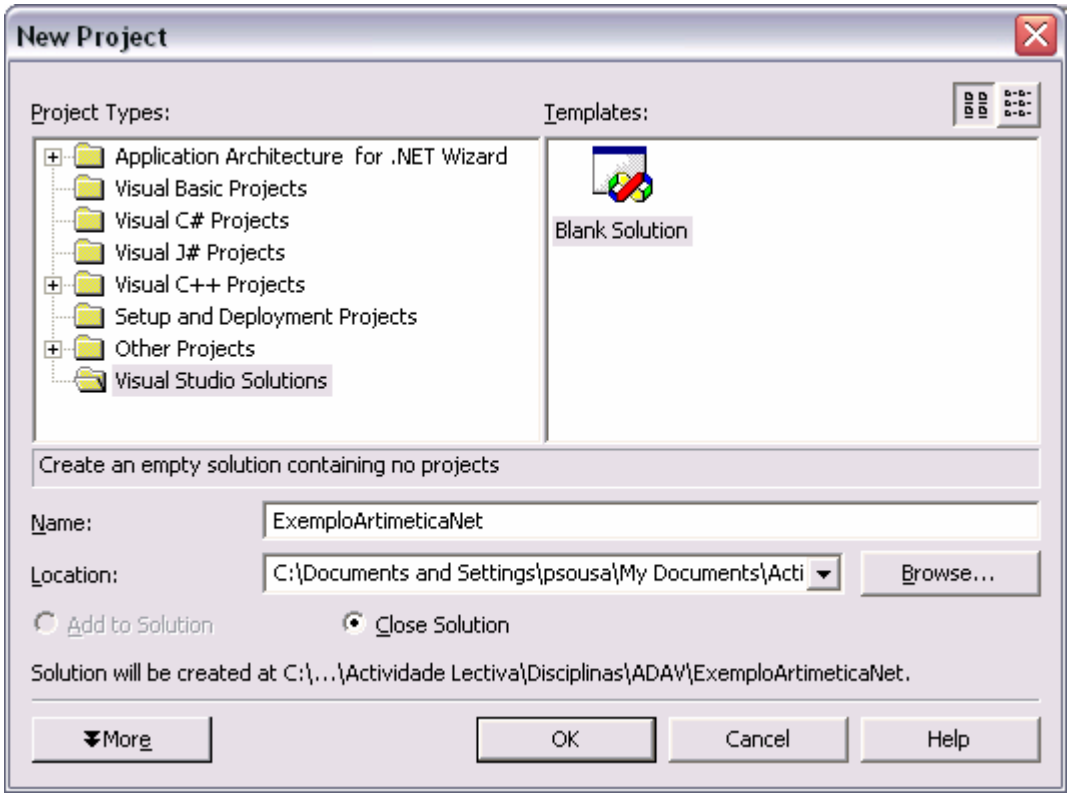

Figura 14 – criar uma "solução" sem projectos iniciais

4.3 Class Library

À solução anterior adicionar um projecto para o nosso componente (*Class Library*) usando File → New → Project.

NOTA: *Podem encontrar regras de codificação .net (ex., nomenclatura de classes) para* class libraries *em* http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp*. Associado a este conjunto de regras existe uma ferramenta intitulada FxCop (disponível em* http://www.gotdotnet.com/team/fxcop/*) que executa a validação de um* assembly *.net no que toca às regras de codificação e algumas boas práticas.*

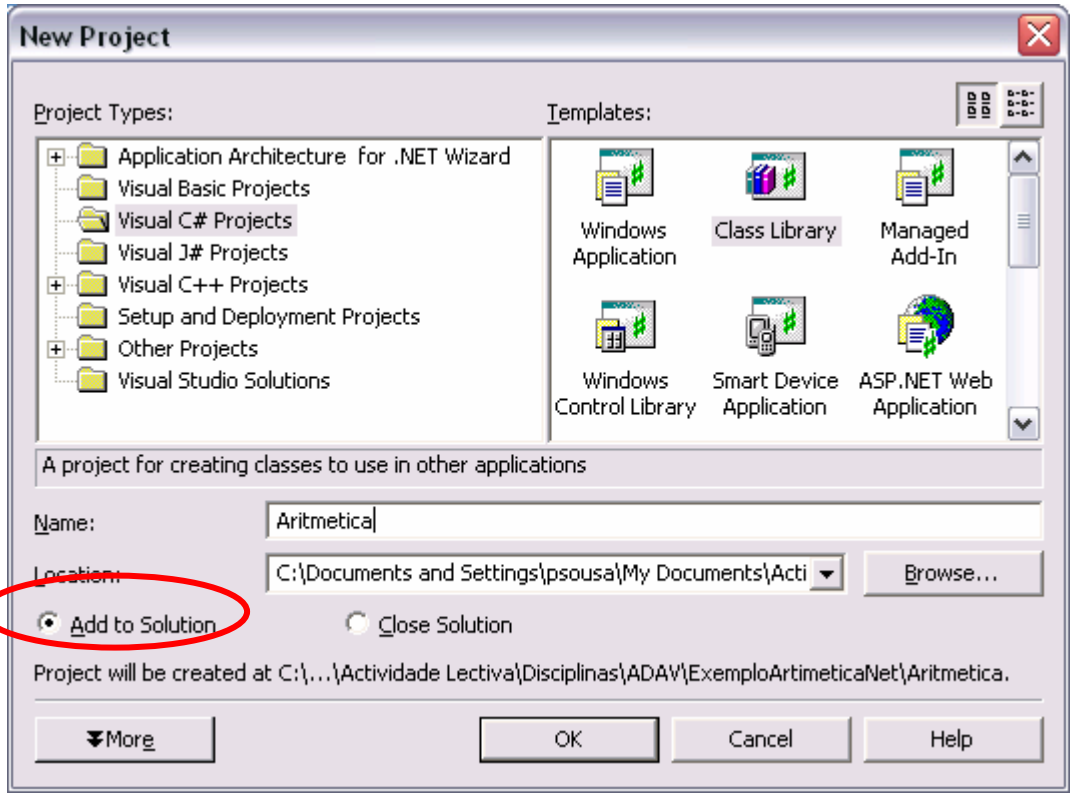

Figura 15 – criar um projecto tipo Class Library em C# e adicionar a uma solução existente

O Visual Studio vai criar o projecto e criar uma série de ficheiros pré-definidos. No “Solution Explorer” remover o ficheiro class1.cs.

Em seguida iremos especificar o contrato (a interface) do nosso componente, ou seja, que operações e que parâmetros recebe cada operação estão disponíveis para a execução de um serviço. No nosso exemplo o serviço que pretendemos fornecer é o de cálculos aritméticos básicos. Para tal, vamos adicionar uma interface ao projecto; clicando com o botão do lado direito do rato em cima do projecto, escolher a opção Add → Add New Item e criar uma classe chamada `IAritmetica`.

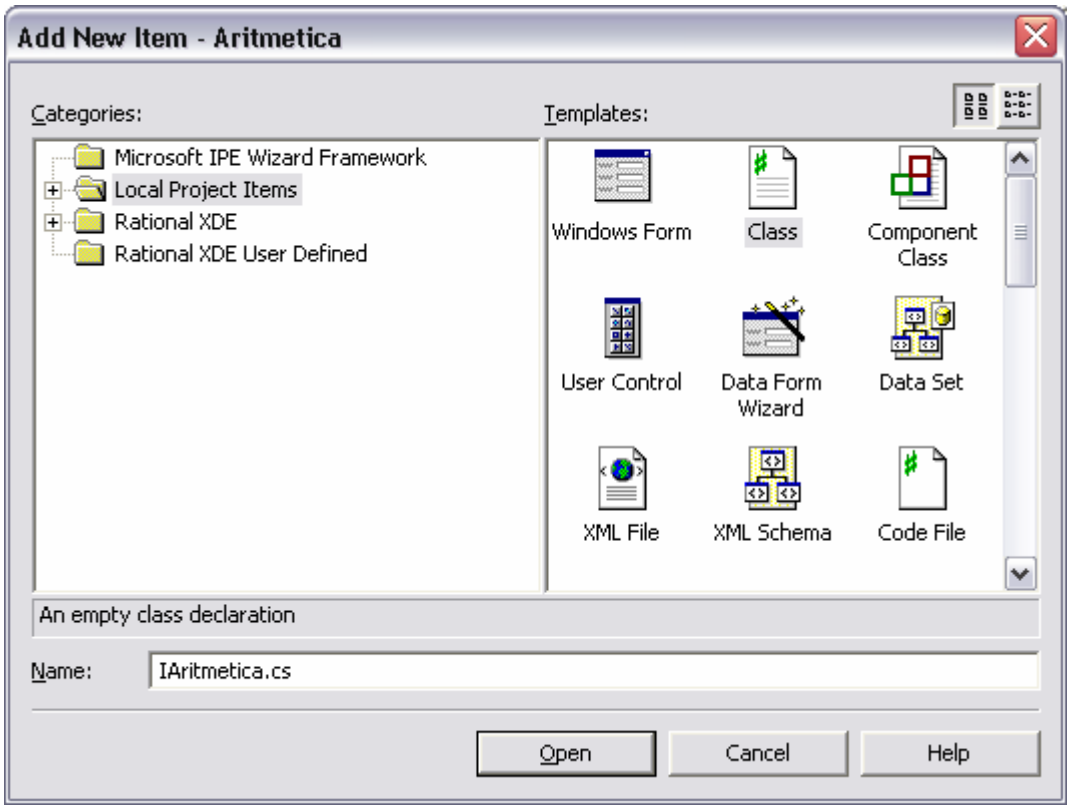

Figura 16 – adicionar uma classe a um projecto

Em seguida substituir a palavra `class` por `interface` e retirar o construtor gerado por omissão.

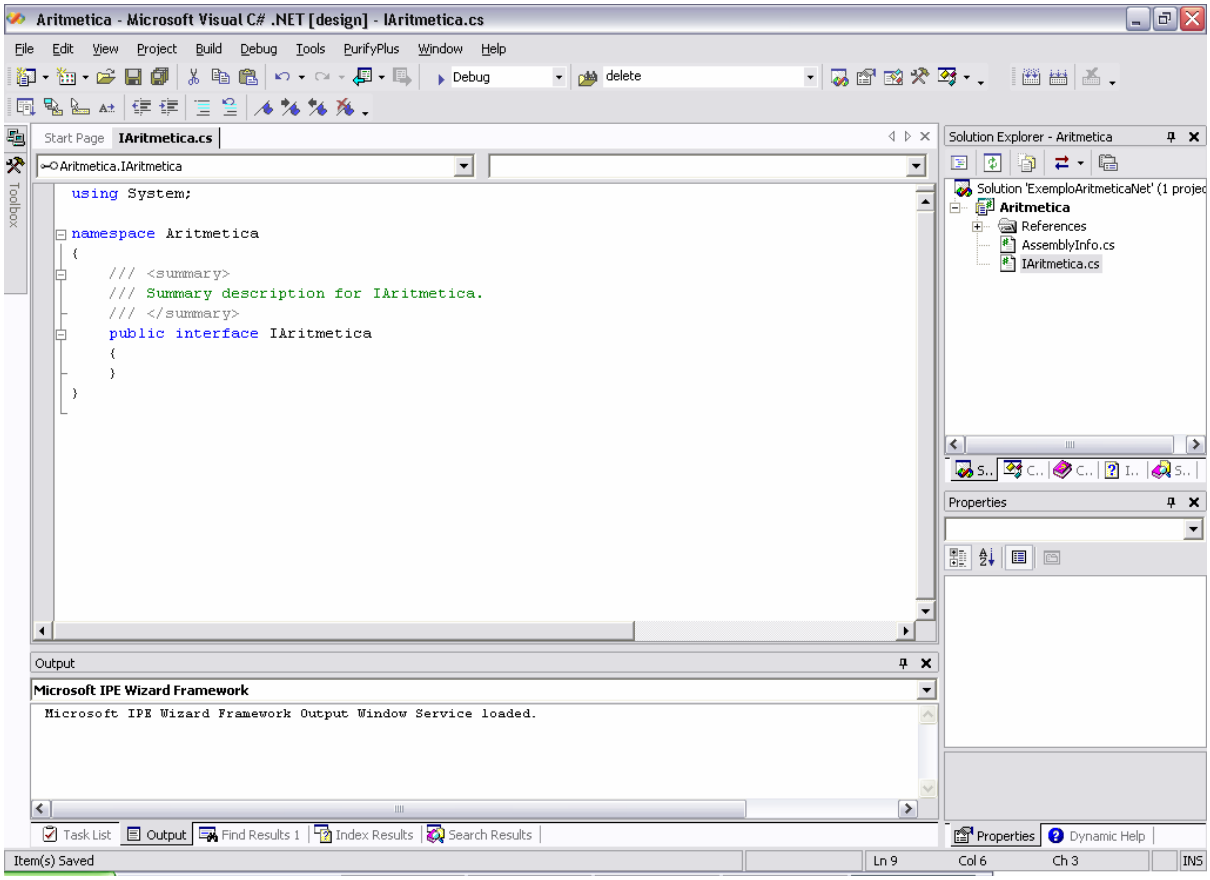

Figura 17 – esqueleto da interface

Mudando de vista no *solution explorer* para se ver as classes existentes no projecto (*class explorer* – ver Figura 18), adicionar à interface um método usando o botão do lado direito do rato (Figura 19).

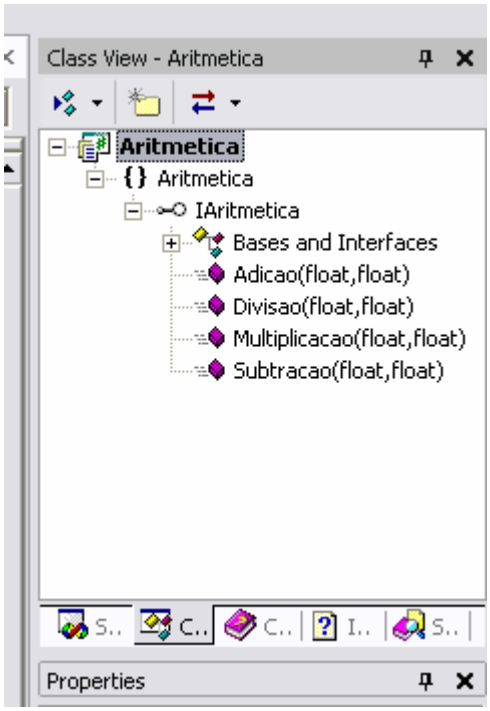

Figura 18 – vista de estrutura de classes no class explorer

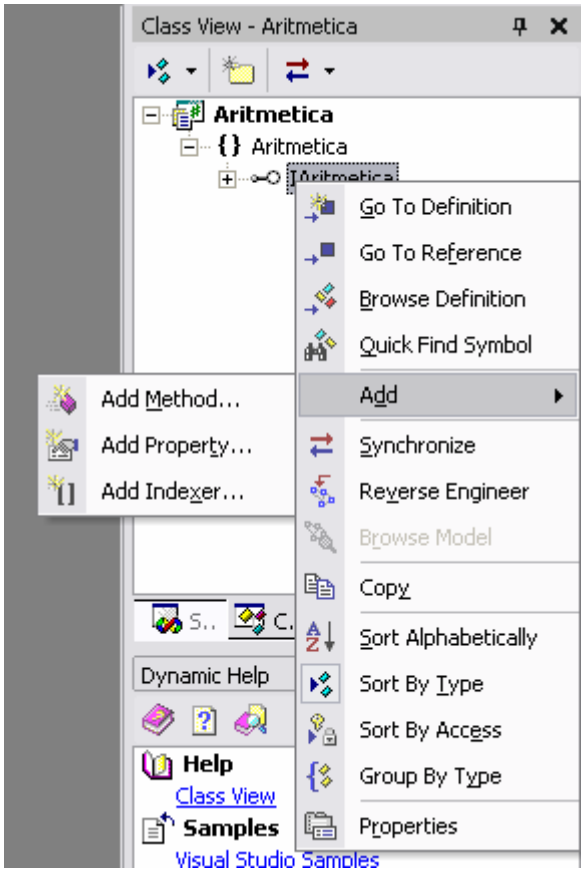

Figura 19 – adicionar um método a uma interface via wizard (passo 1)

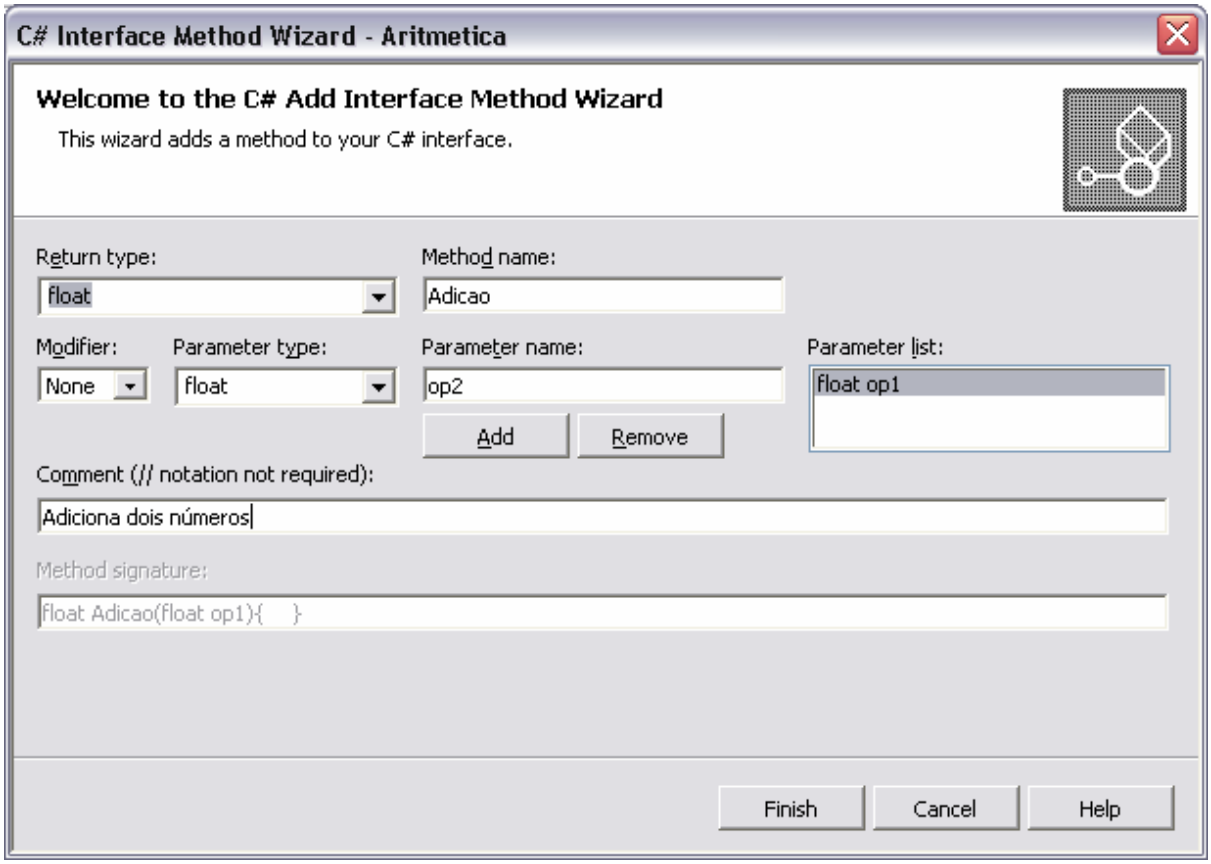

Figura 20 – adicionar um método a uma interface via wizard (passo 2)

O *wizard* vai automaticamente criar o método no ficheiro fonte.

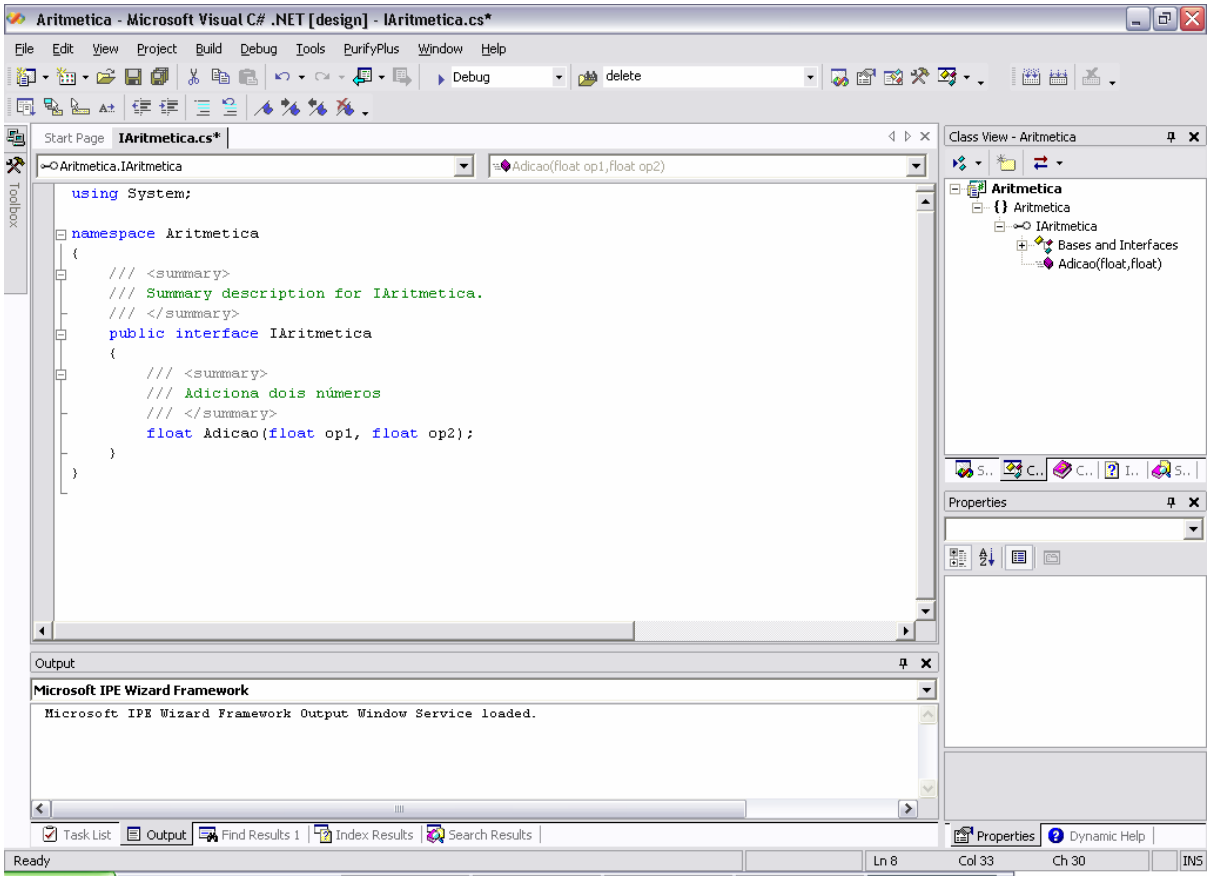

Figura 21 – código gerado automaticamente pelo wizard de adição de métodos

Da mesma forma adicionar os restantes métodos para a subtracção, divisão e multiplicação.

É também possível escrever os métodos directamente no editor de texto sem usar o wizard.

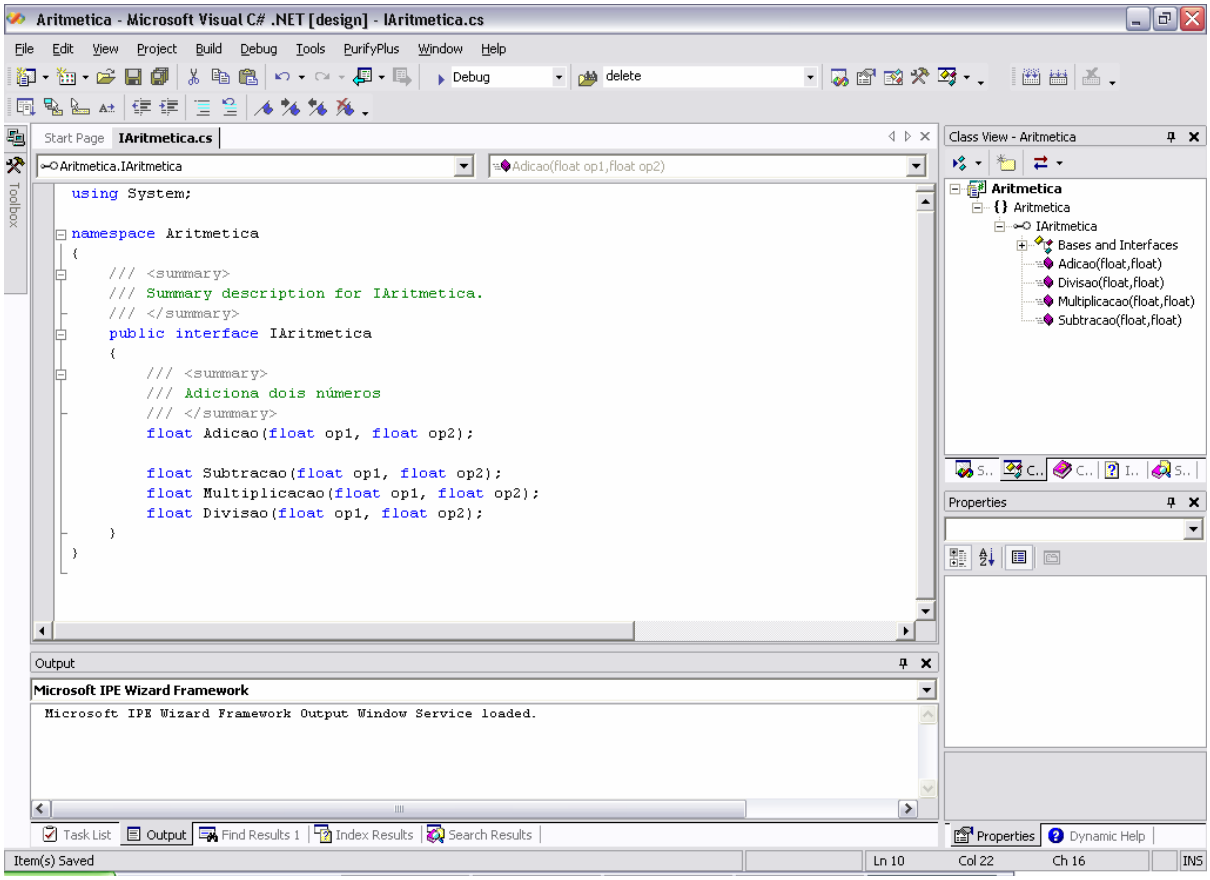

Figura 22 – interface para o serviço de aritmética

Em seguida vamos criar o prestador de serviços propriamente dito, ou seja, a implementação dos métodos definidos na interface anterior. Para tal, vamos adicionar uma classe ao projecto, seleccionando no *class explorer* o projecto "Aritmetica" e com o botão do lado direito do rato em cima do projecto, escolher a opção Add → Add Class.

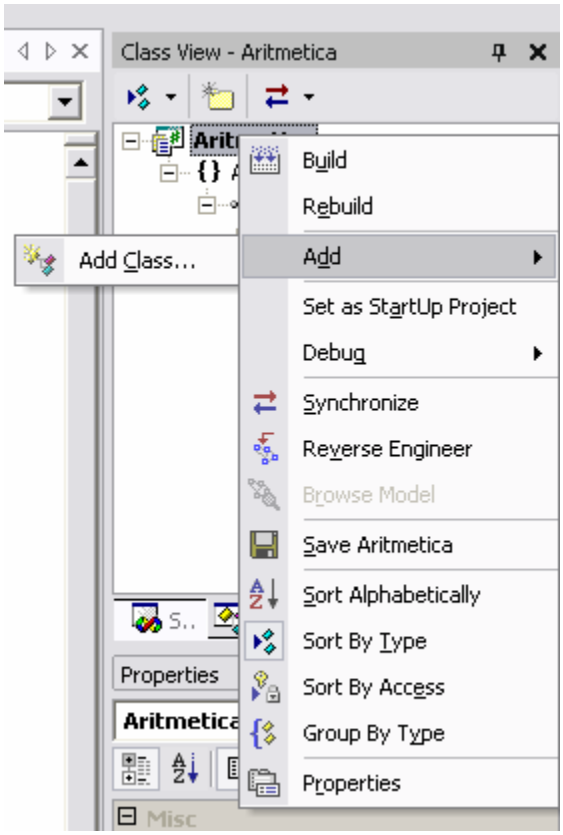

Figura 23 – adicionar uma classe via wizard a partir do class explorer (passo 1)

Irá aparecer o *wizard* de criação de classes (Figura 24) no qual indicaremos o nome da classe a criar e uma pequena descrição.

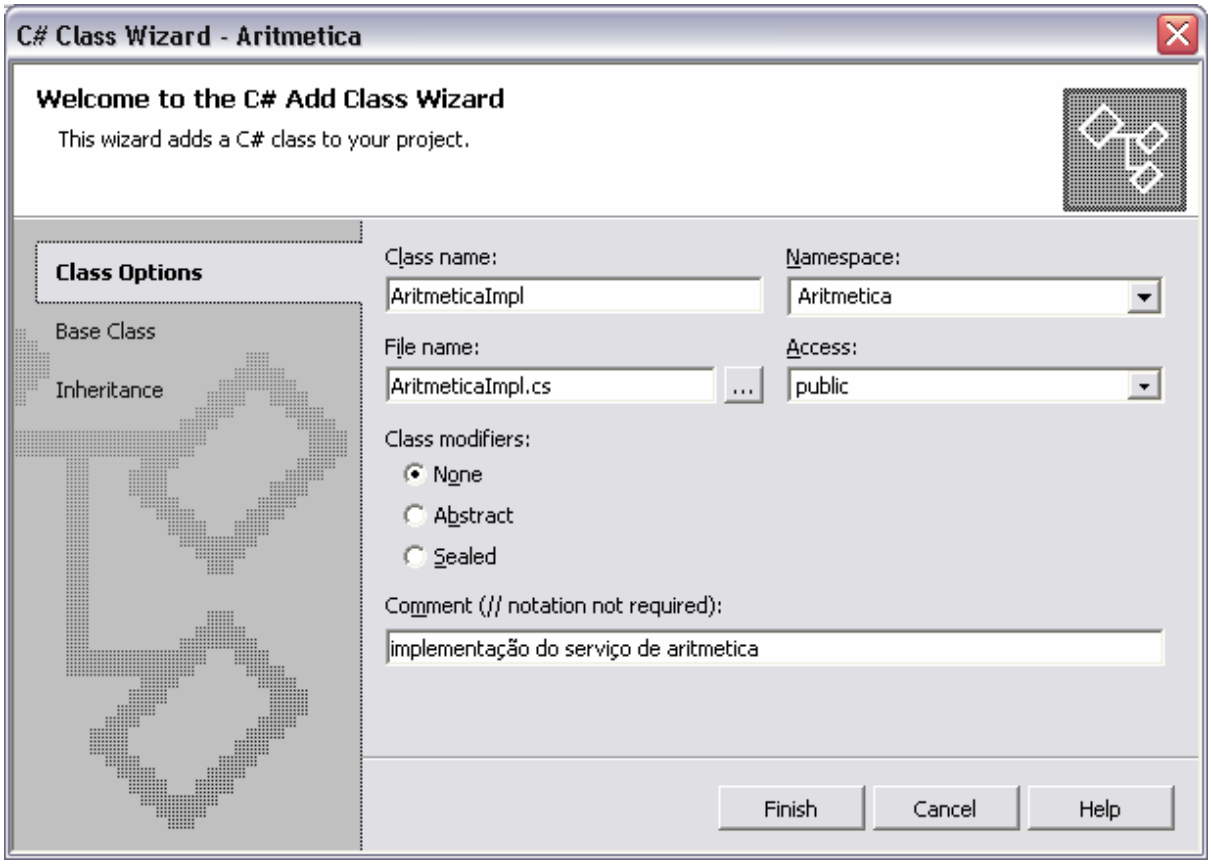

Figura 24 – adicionar uma classe via wizard a partir do class explorer (passo 2)

Em seguida iremos indicar características de herança desta classe seleccionando o separador “Inheritance” do lado esquerdo da janela do *wizard* (Figura 25).

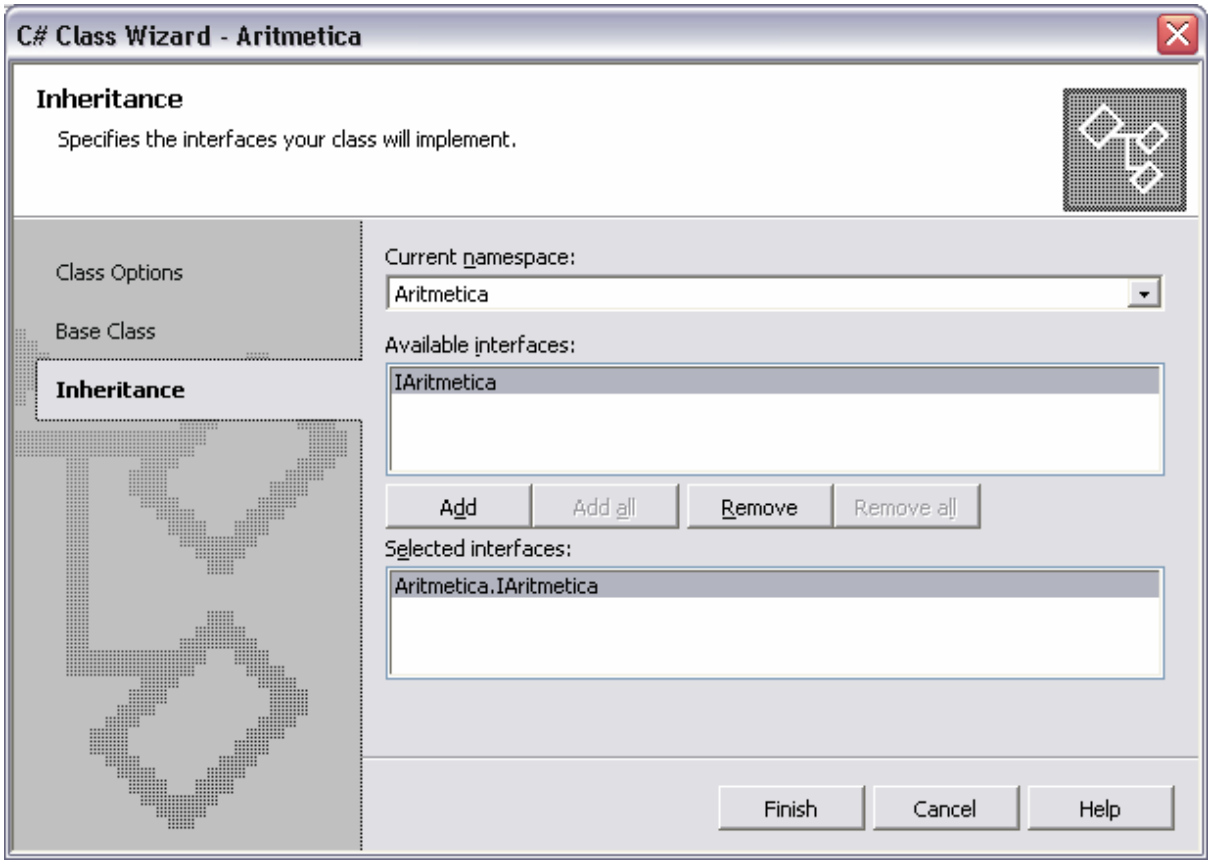

Figura 25 – adicionar uma classe via wizard a partir do class explorer (passo 3)

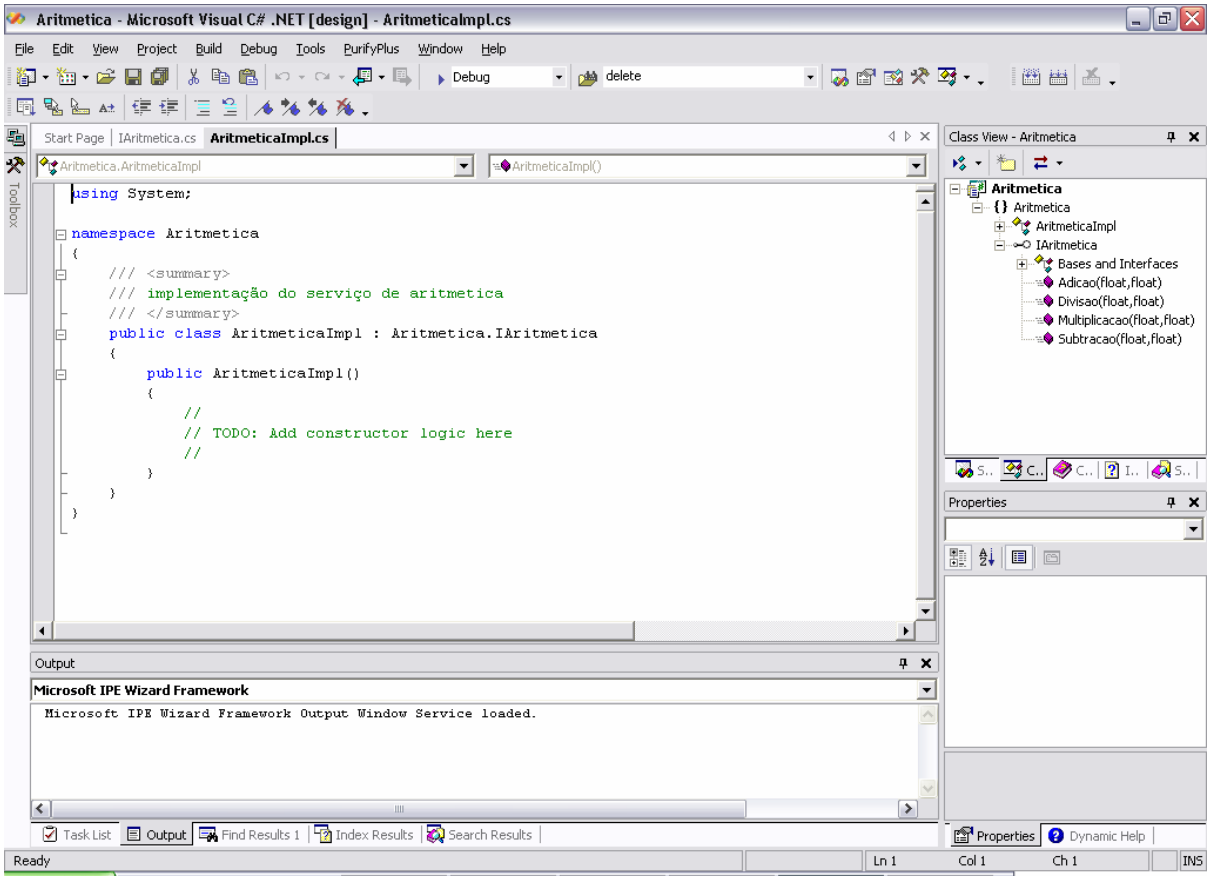

Figura 26 – classe gerada para implementação do serviço

Vamos em seguida implementar os métodos definidos na interface. Para isso no *class explorer* escolher das classes bases e interfaces da classe `AritmeticaImpl`, o item com a interface `IAritmetica` e com o botão do lado direito escolher Add → Implement Interface (Figura 27).

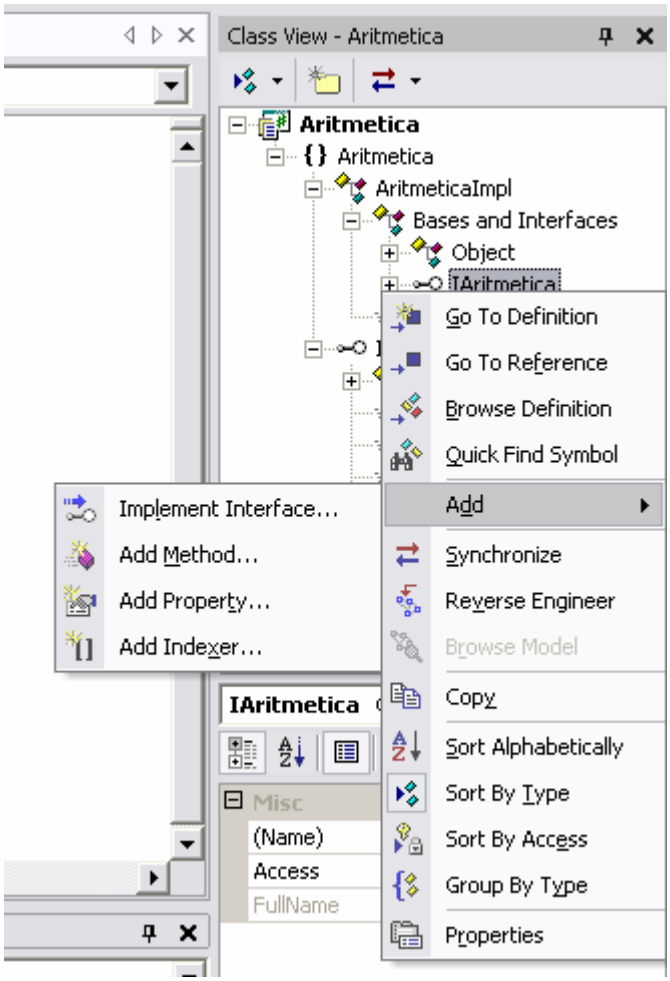

Figura 27 – implementar um interface

O Visual Studio gerou o esqueleto de código associado à interface `IAritmetica` e alterou o ficheiro fonte da classe `ArtimeticaImpl` conforme se pode ver na figura seguinte.

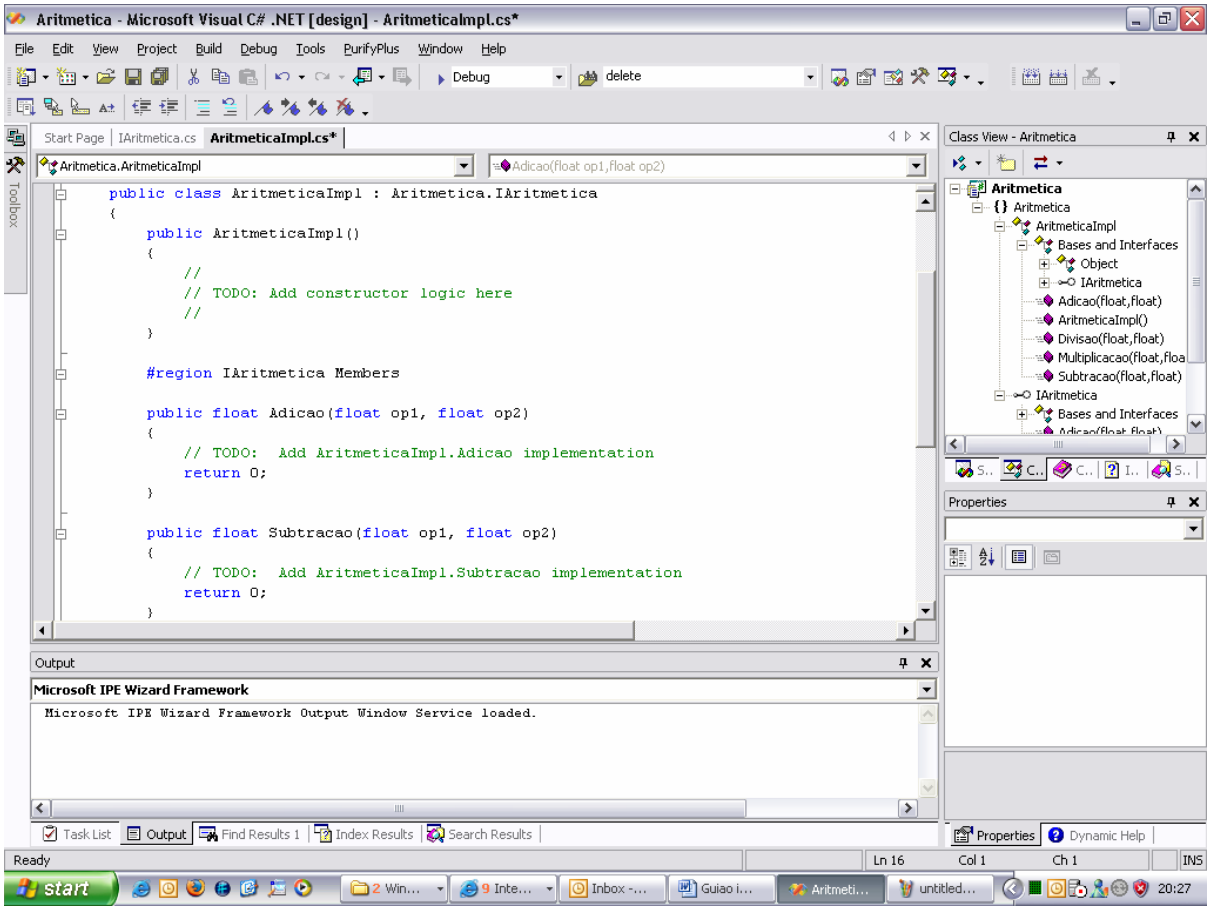

Figura 28 – classe de implementação do serviço com esqueleto dos métodos da interface

Colocar o seguinte código na implementação dos métodos:

```
public float Adicao(float op1, float op2)
{
   return op1+op2;
}

public float Subtracao(float op1, float op2)
{
   return op1-op2;
}

public float Multiplicacao(float op1, float op2)
{
   return op1*op2;
}

public float Divisao(float op1, float op2)
{
   if (op2 != 0)
      return op1/op2;
   else
      throw new ApplicationException("Tentativa de efectuar uma Divisão
por zero ");
}
```

De notar que na implementação do método `Divisao()` recorremos a excepções para indicar ao utilizador desta classe situações anómalas. Neste caso utilizamos a classe predefinida `ApplicationException`, embora fosse possível definir as nossas próprias excepções[8].

Compilar (não deve dar erros).

4.4 Consola de teste

Vamos agora criar um novo projecto para testar o componente anterior. Usar File → New e escolher um novo projecto garantindo que se adiciona esse projecto à solução actual.

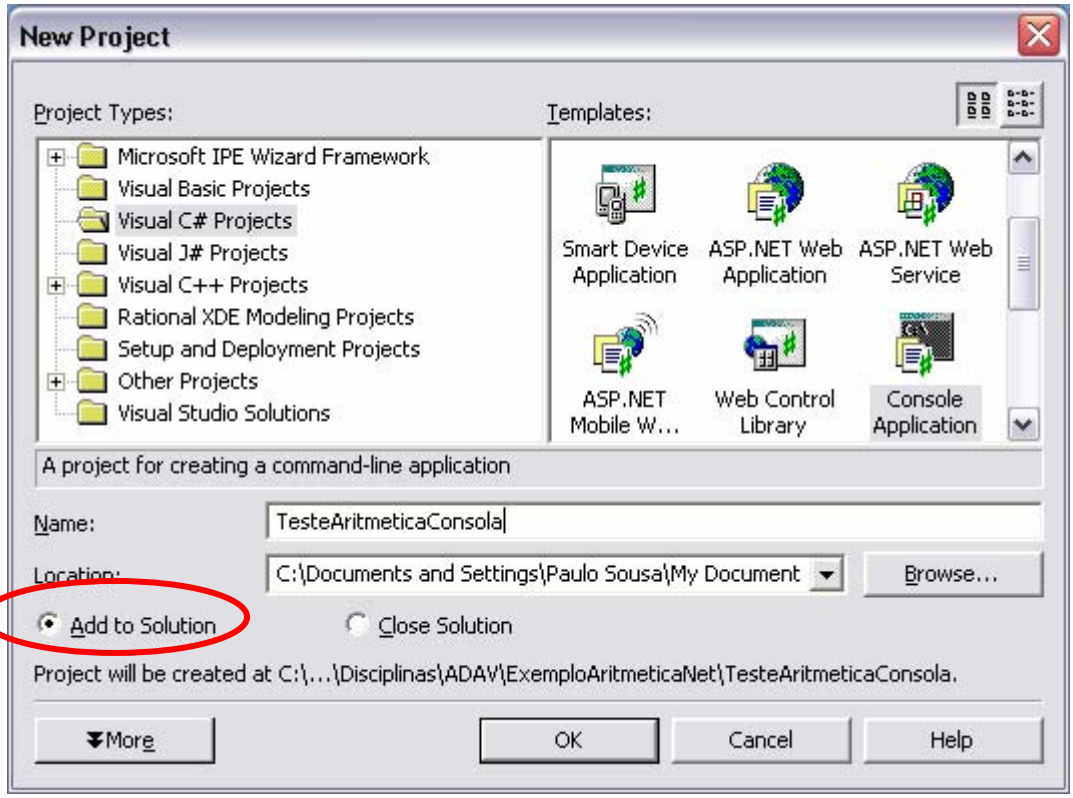

Figura 29 – criar um novo projecto de consola em C#

Para que possamos usar o componente anterior temos que adicionar uma referência ao componente. Para isso usamos o botão do lado direito do rato sobre o item “references” no *solution explorer*.

[8] Para mais informações sobre criação, lançamento (*throw*) e tratamento (*try-catch*) de excepções ver http://msdn.microsoft.com/library/en-us/cpguide/html/cpconhandlingthrowingexceptions.asp?frame=true

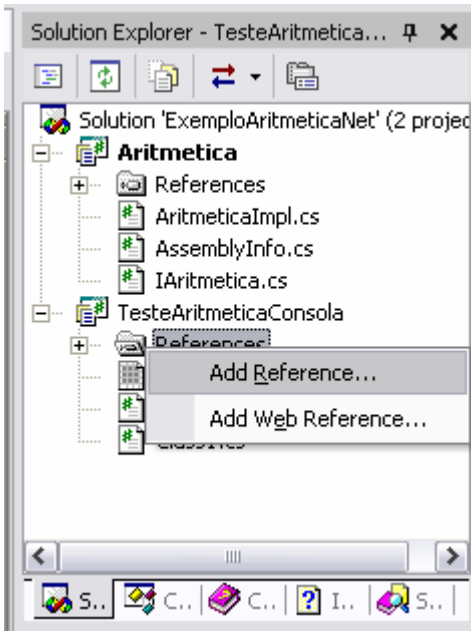

Figura 30 - Adicionar uma referência a outro projecto (passo 1)

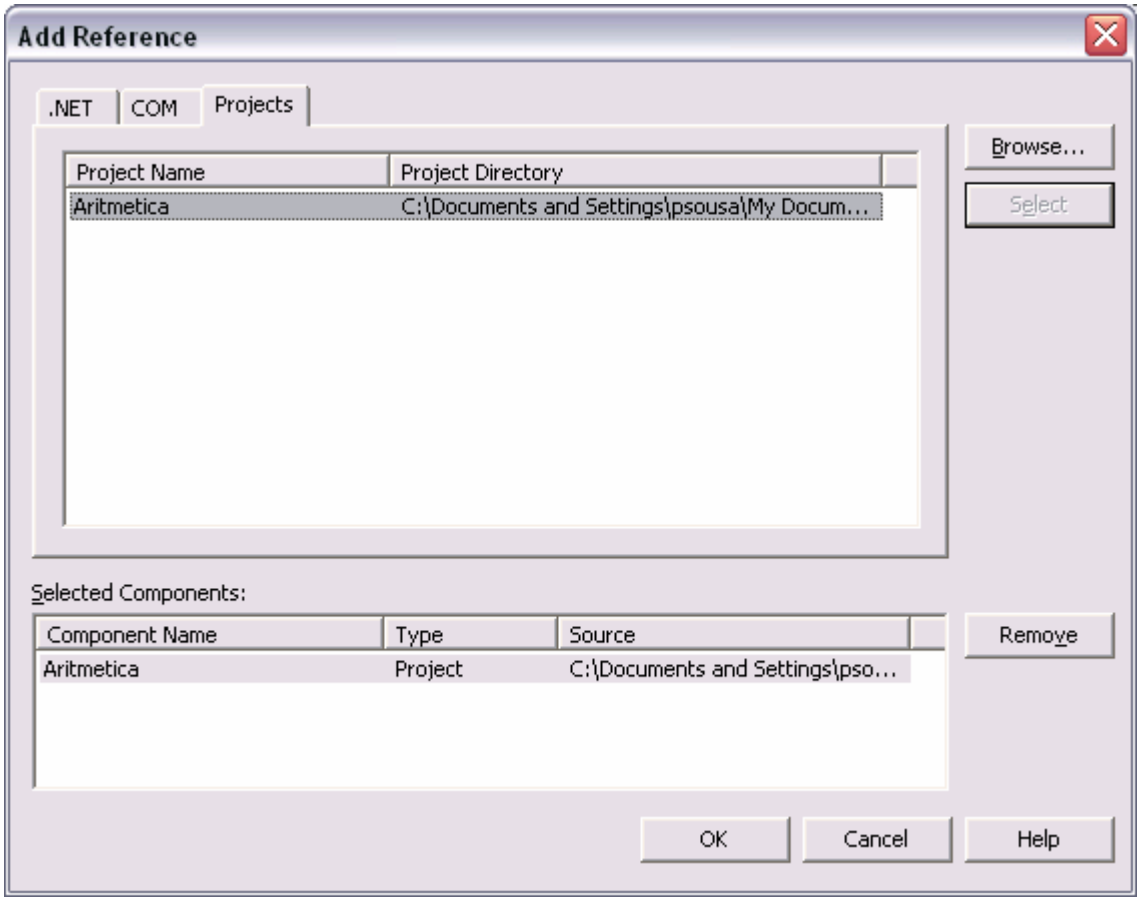

Figura 31 – Adicionar uma referência a outro projecto (passo 2)

No ficheiro class1.cs criado automaticamente, procurar o método `Main()` e colocar o seguinte código:

```
static void Main(string[] args)
{
   int op1 = 5;
   int op2 = 7;

   //declarar objecto para o serviço que pretendemos - interface
   Aritmetica.IAritmetica comp;

   //criar objecto com implementação do serviço
   comp = new Aritmetica.AritmeticaImpl();

   //invocar o serviço pretendido
   Console.WriteLine("Adição({0}, {1}) => {2}", op1, op2,
                              comp.Adicao(op1, op2)
                     );
}
```

De notar que a variável `comp` é do tipo da interface `IAritmetica` e não do tipo da classe `AritmeticaImpl`. Embora fosse possivel, e o funcionamento do programa seria idêntico, declarar a variável `comp` do tipo `AritmeticaImpl`, conceptualmente o contrato entre a aplicação e o componente é expresso pela sua interface; sendo assim, é mais correcto e expressivo manipular o componente através da interface. Uma vantagem adicional deste método é permitir com alguma facilidade a substituição da implementação da interface (por ex., utilizar o componente de um outro fabricante), visto que apenas a linha com a criação da classe que implementa o serviço necessita ser alterada.

O método `WriteLine()` permite a indicação de parâmetros indicados pelos números entre chavetas, para uma string de formatação[9].

Para testar é necessário indicar ao Visual Studio qual o projecto a executar quando constam vários projectos na mesma solução. Para tal, com o botão do lado direito em cima do projecto no *solution explorer*, escolher “Set as startup project” (Figura 32).

[9] Para mais informação sobre o método `WriteLine()` consultar http://msdn.microsoft.com//library/en-us/cpref/html/frlrfsystemconsoleclasswritelinetopic15.asp; para informação sobre as strings de formatação consultar http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcompositeformatting.asp.

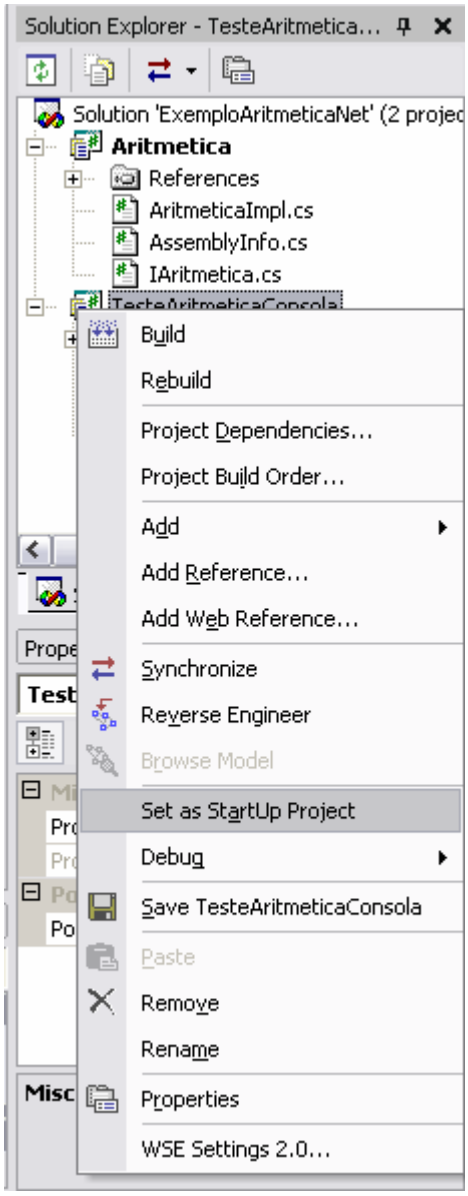

Figura 32 – seleccionar um projecto como projecto inicial para execução

Compilar e testar.

Figura 33 – aspecto geral da aplicação de consola

4.5 Aplicação Winforms de teste

Vamos agora criar outro tipo de cliente, uma aplicação windows neste caso em Visual Basic.net. usando File → New adicionar um novo projecto à solução.

Este projecto utiliza a linguagem de programação Visual Basic .net e não C# para demonstrar que é possível utilizar componentes escritos numa dada linguagem de programação a partir de clientes (programas ou outros componentes) escrita noutra linguagem de programação.

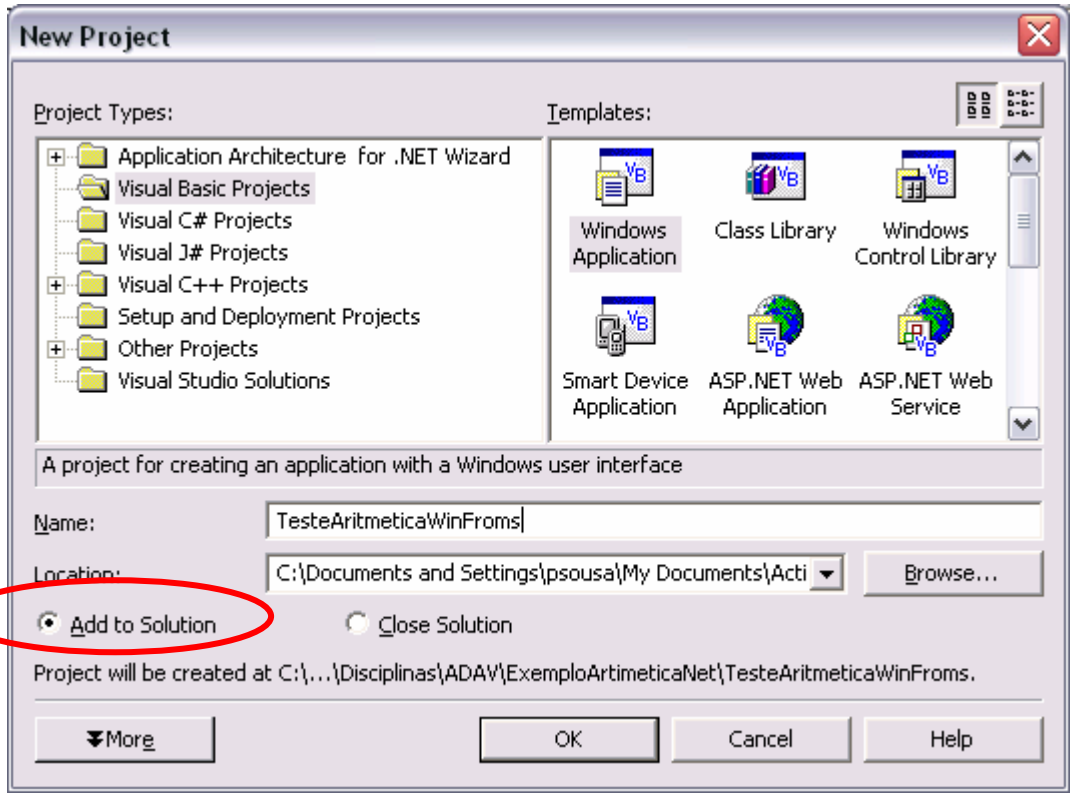

Figura 34 – criar um projecto WinForms em VB.net

Adicionar a referência ao componente como foi feito no projecto anterior.

Usando o editor visual construir o formulário da aplicação para que se pareça com o da imagem seguinte (utilize a janela “Properties” para atribuir os valores das propriedades dos controlos de acordo com a Tabela 2).

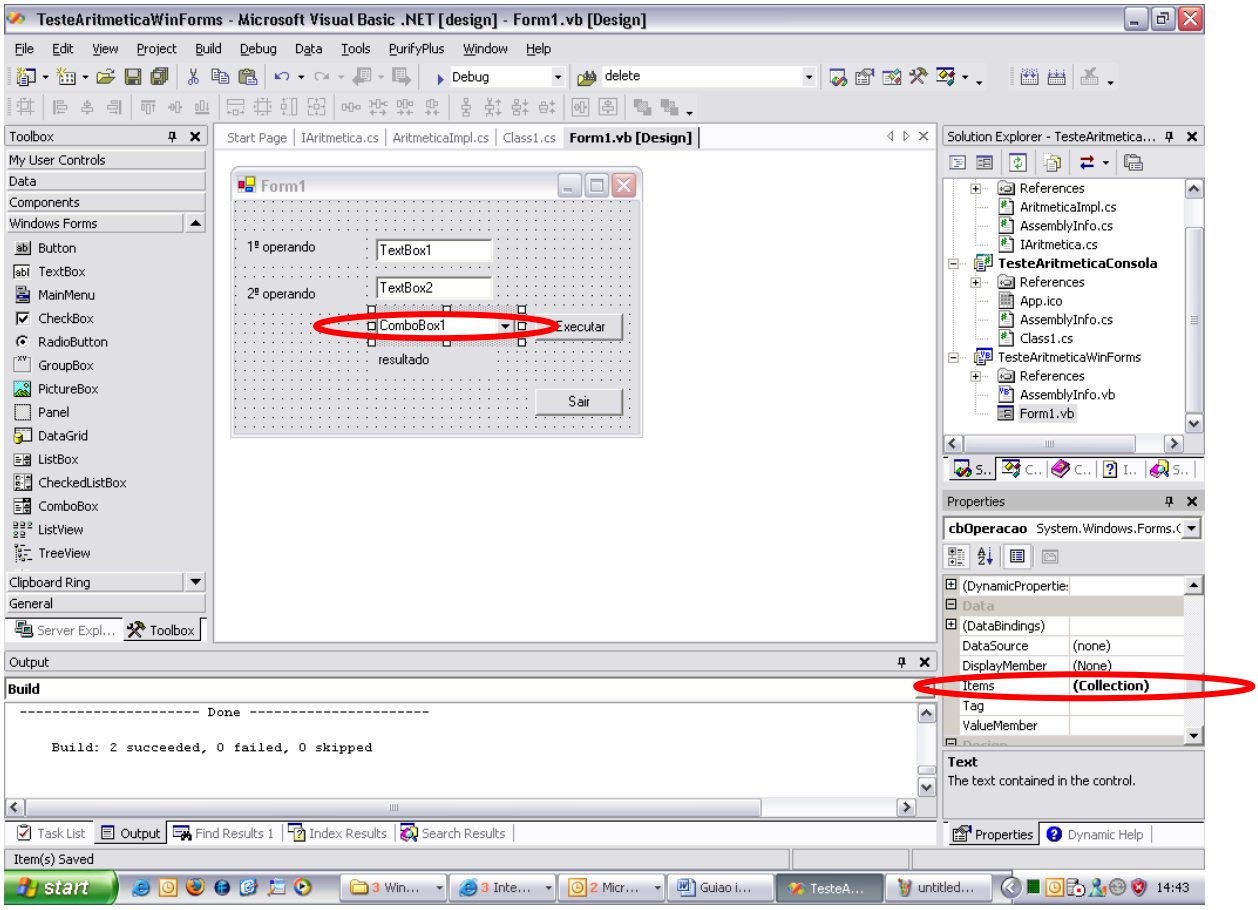

Figura 35 – adicionar itens alfanuméricos a uma ListBox ou ComboBox (passo 1)

Tabela 2 – propriedades dos controlos da aplicação WinForms

Elemento	Propriedade	Valor	Elemento	Propriedade	Valor
1ª text box	`Name`	txtOp1	2º botão	`Name`	btnSair
1ª text box	`Text`		2º botão	`Text`	Sair
2ª text box	`Name`	txtOp2	1º label	`Text`	1º operando
2ª text box	`Text`		2º label	`Text`	2º operando
combobox	`Name`	cbOperacao	3ª label	`Name`	lblResultado
1º botão	`Name`	btnExecutar	3º label	`Text`	Resultado
1º botão	`Text`	Executar			

A `ComboBox` pode ser preenchida com valores em tempo de desenho usando a propriedade `Collection`.

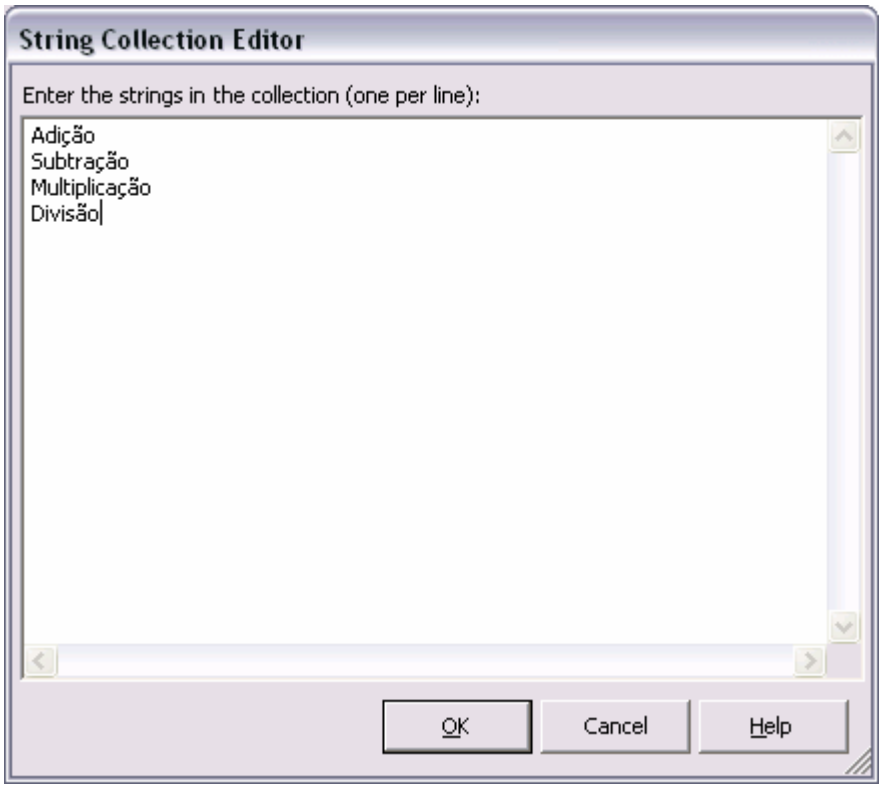

Figura 36 – adicionar itens alfanuméricos a uma ListBox ou ComboBox (passo 2)

Para testar a adição, criar um método de evento[10] associado ao botão de executar. Para isso efectuar um duplo *click* no botão e no método gerado colocar o seguinte código:

```
Private Sub btnExecutar_Click(ByVal sender As System.Object,
                              ByVal e As System.EventArgs)
                            Handles btnExecutar.Click
Dim oper1 As Single
Dim oper2 As Single

   'declarar variavel
   Dim comp as Aritmetica.IAritmetica

   oper1 = Single.Parse(txtOp1.Text)
   oper2 = Single.Parse(txtOp2.Text)

   'criar objecto com implementação do serviço
   comp = New Aritmetica.AritmeticaImpl
```

[10] Para mais informações sobre tratamento de eventos em aplicações que utilizam Win Forms consultar http://msdn.microsoft.com/library/en-us/vbcon/html/vbconeventhandling.asp; para informação geral sobre o mecanismo de eventos na plataforma .net consultar http://msdn.microsoft.com/library/en-us/vbcon/html/vborieventhandlers.asp.

```
   lblResultado.Text = comp.Adicao(oper1, oper2)
End Sub
```

Vamos também inicializar o formulário criando o método que trata o evento de carregamento do formulário (indicado pela palavra reservada `Handles` e pelo evento `Load`), fazendo um duplo *click* em qualquer área livre o mesmo e colocando o seguinte código na função gerada.

```
Private Sub Form1_Load(ByVal sender As System.Object,
                       ByVal e As System.EventArgs)
                    Handles MyBase.Load
   ´colocar como seleccionado a 1ª opção da listbox
   cbOperacao.SelectedIndex = 0
End Sub
```

Finalmente vamos colocar código associado ao botão de saída da aplicação para fechar a janela e sair da aplicação.

```
Private Sub btnSair_Click(ByVal sender As System.Object,
                       ByVal e As System.EventArgs)
                       Handles btnSair.Click
   Me.Close()
End Sub
```

Seleccionar o projecto “TesteAritmeticaWinForms” como projecto *start up*, compilar e testar.

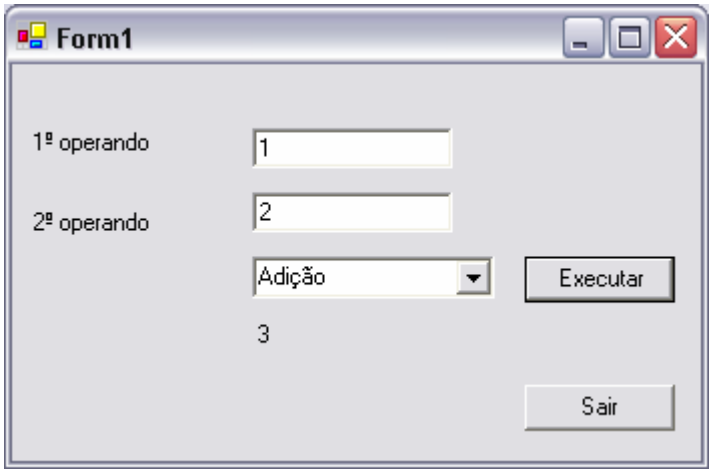

Figura 37 – aspecto geral da aplicação windows

Vamos agora colocar em funcionamento as restantes opções:

```
Private Sub btnExecutar_Click(ByVal sender As System.Object,
                        ByVal e As System.EventArgs)
                        Handles btnExecutar.Click
   Dim oper1 As Single
   Dim oper2 As Single
```

```
    'declarar variavel
    Dim comp as Aritmetica.IAritmetica

    oper1 = Single.Parse(txtOp1.Text)
    oper2 = Single.Parse(txtOp2.Text)

    'criar objecto com implementação do serviço
    comp = New Aritmetica.AritmeticaImpl

    If cbOperacao.SelectedItem = "Adição" Then
       lblResultado.Text = comp.Adicao(oper1, oper2)
    End If
    If cbOperacao.SelectedItem = "Subtracção" Then
       lblResultado.Text = comp.Subtracao(oper1, oper2)
    End If
    If cbOperacao.SelectedItem = "Multiplicação" Then
       lblResultado.Text = comp.Multiplicacao(oper1, oper2)
    End If
    If cbOperacao.SelectedItem = "Divisão" Then
       lblResultado.Text = comp.Divisao(oper1, oper2)
    End If
End Sub
```

Compilar e testar.

Como se pode ver, no caso da divisão, se tentarmos dividir por zero o programa gera uma excepção. Podemos programaticamente controlar essa excepção fazendo a seguinte alteração:

```
If cbOperacao.SelectedItem = "Divisão" Then
   Try
      lblResultado.Text = comp.Divisao(oper1, oper2)
   Catch ex As ApplicationException
      System.Windows.Forms.MessageBox.Show("Não pode dividir por Zero.")
   End Try
End If
```

Compilar e testar.

Adicionalmente, pode também ser gerada uma excepção caso se introduzam caracteres não numéricos nas caixas de texto (ou mesmo se a caixa de texto estiver vazia) quando se tenta fazer o `Single.Parse()`. Para corrigir esse problema devemos alterar o código do método para o seguinte:

```
Private Sub btnExecutar_Click(ByVal sender As System.Object,
                              ByVal e As System.EventArgs)
                          Handles btnExecutar.Click
Dim oper1 As Single
Dim oper2 As Single
```

```
'declarar variavel
Dim comp As Aritmetica.IAritmetica

   'criar objecto com implementação do serviço
   comp = New Aritmetica.AritmeticaImpl

   Try
      oper1 = Single.Parse(txtOp1.Text)
      oper2 = Single.Parse(txtOp2.Text)

      If cbOperacao.SelectedItem = "Adição" Then
         lblResultado.Text = comp.Adicao(oper1, oper2)
      End If
      If cbOperacao.SelectedItem = "Subtracção" Then
         lblResultado.Text = comp.Subtracao(oper1, oper2)
      End If
      If cbOperacao.SelectedItem = "Multiplicação" Then
         lblResultado.Text = comp.Multiplicacao(oper1, oper2)
      End If
      If cbOperacao.SelectedItem = "Divisão" Then
         Try
            lblResultado.Text = comp.Divisao(oper1, oper2)
         Catch ex As ApplicationException
            System.Windows.Forms.MessageBox.Show("Não pode dividir por
Zero.")
         End Try
      End If
   Catch ex As FormatException
      System.Windows.Forms.MessageBox.Show("Por favor preencha as caixas
de texto apenas com números")
   End Try
End Sub
```

Compilar e testar.

4.6 Aplicação ASP.net de teste

Vamos agora utilizar o nosso componente a partir de uma aplicação web, para isso criar um novo projecto web em C# na solução que temos vindo a usar. E adicionar a referência ao componente tal como foi feito nos passos anteriores.

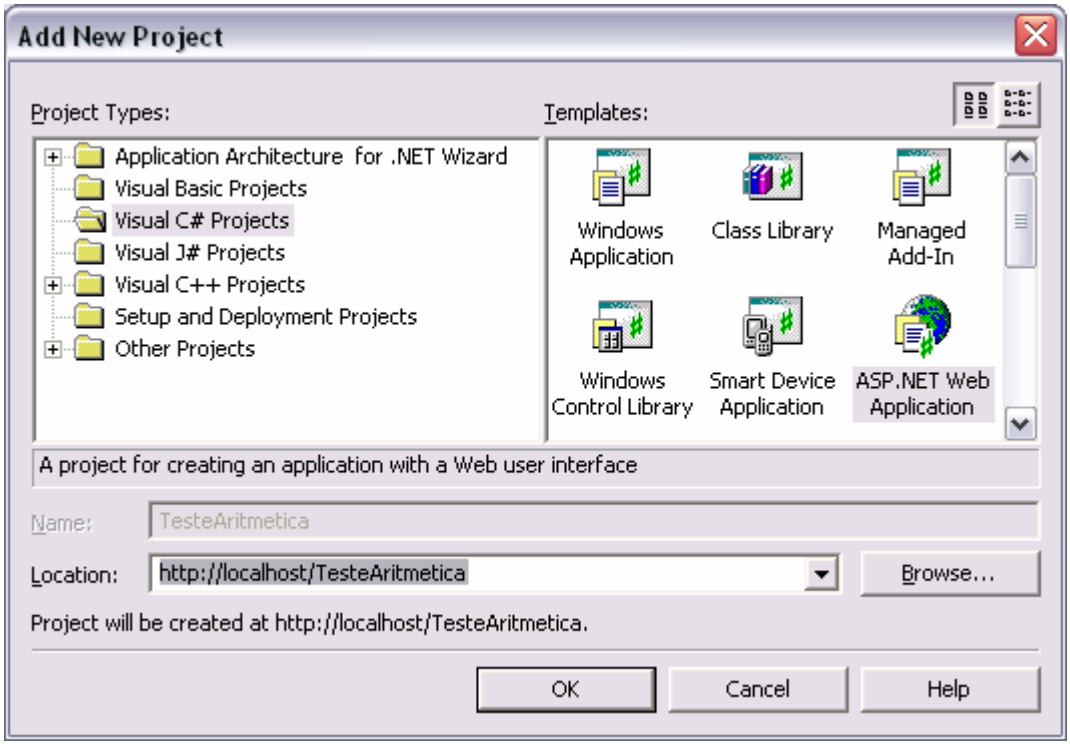

Figura 38 – criar um projecto tipo aplicação web asp.net em C#

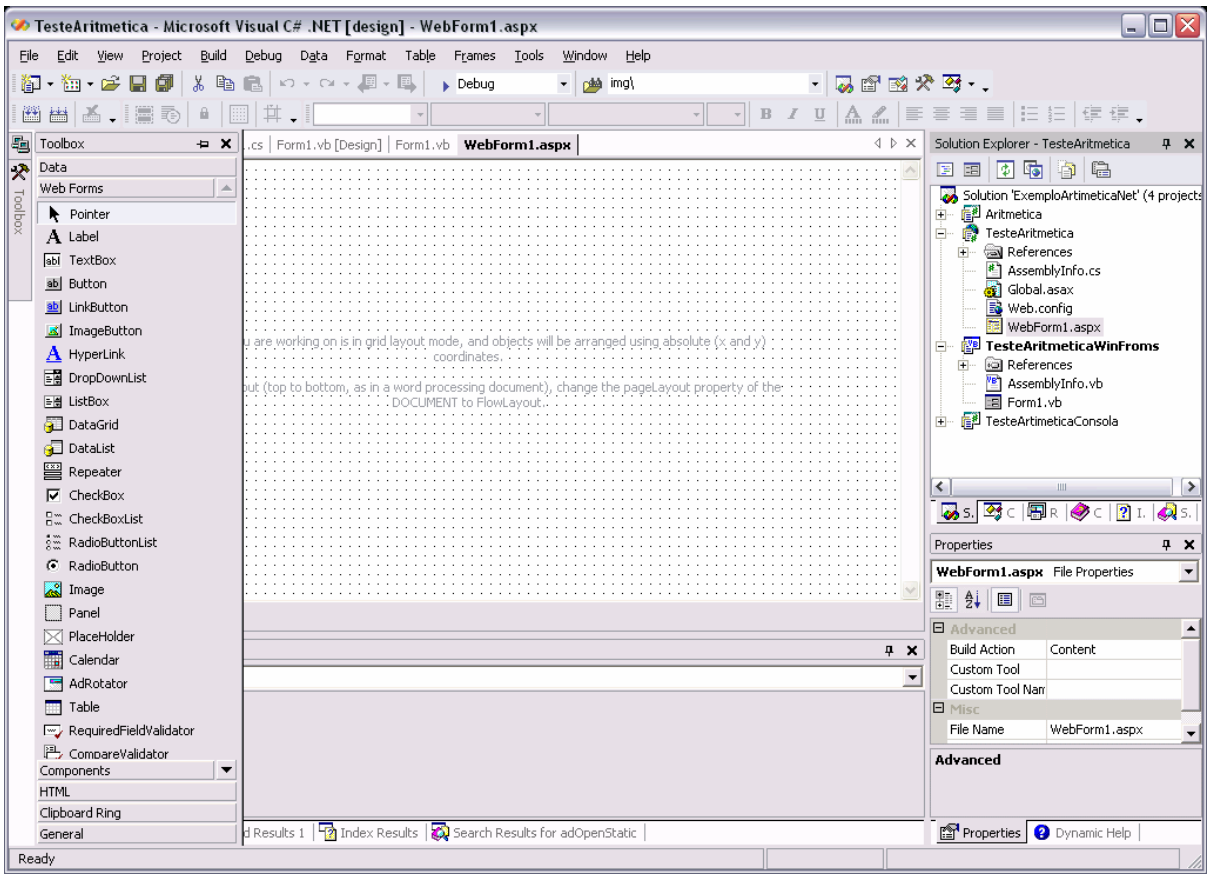

Figura 39 – toolbox de controlos disponíveis para formulários web

Usando a *toolbox* (Figura 39), desenhar o formulário que se pode ver na figura seguinte.

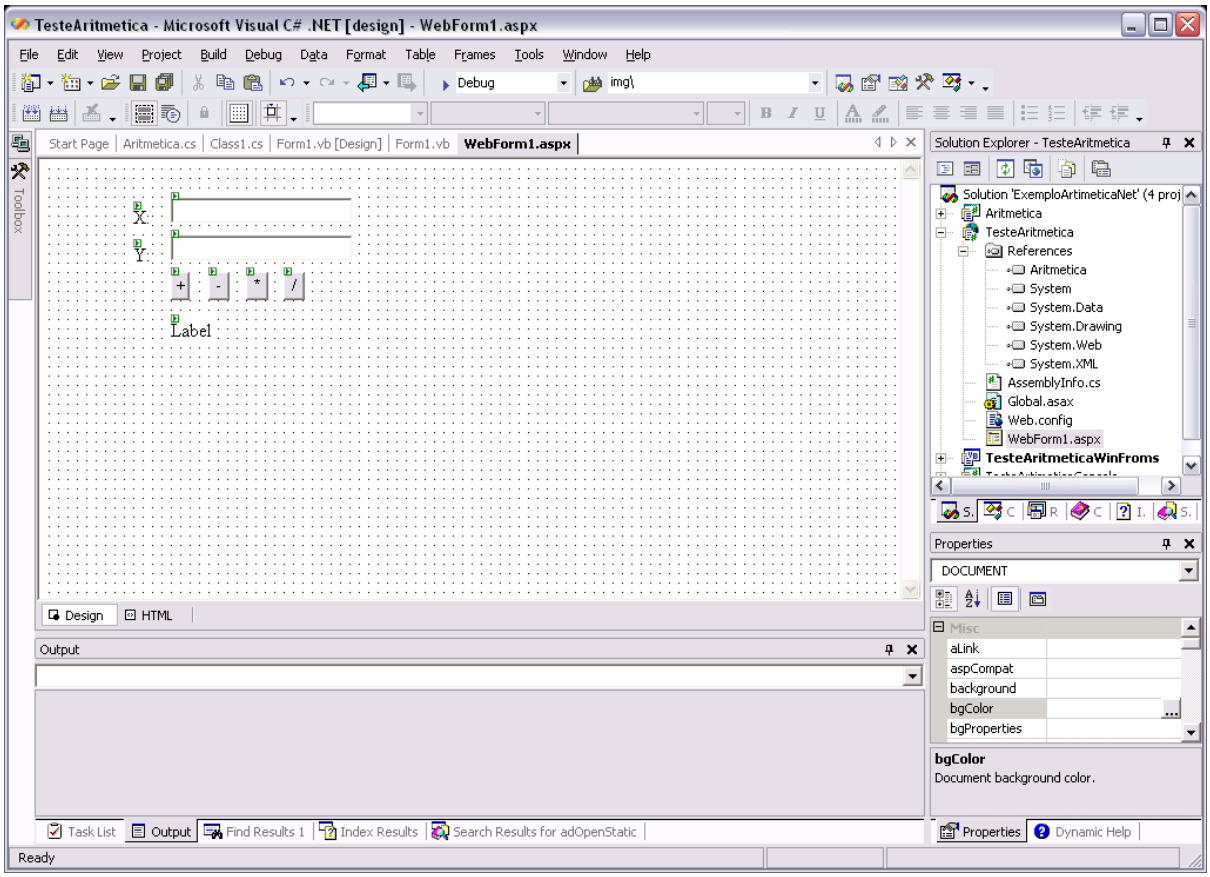

Figura 40 – formulário web a criar para testar componente

Para este exemplo vamos utilizar um botão para cada operação. Para se colocar o botão de adição a funcionar vamos associar um método ao evento de *click* do botão. Para isso fazemos duplo *click* sobre o botão e o Visual Studio automaticamente vai criar um método e associa-lo ao evento *click*. Nesse método colocar o seguinte código.

```
using Aritmetica; // simplifica a escrita do código

private void btnAdicao_Click(object sender, System.EventArgs e)
{
   IAritmetica comp = new AritmeticaImpl();

   try
   {
      lblResultado.Text = comp.Adicao(float.Parse(txtOp1.Text),
                          float.Parse(txtOp2.Text)
                          ).ToString();
   }
   catch (FormatException ex)
   {
      lblResultado.Text = "";
   }
}
```

De reparar que agora em C# é necessário converter os valores da caixa de texto de strings para inteiros usando o método `float.Parse()` e em seguida converter o resultado inteiro para string para ser visualizado usando o método `ToString()`. Neste exemplo também tratamos um caso adicional em que o utilizador tenha colocado caracteres não numéricos nas caixas de texto tratando a excepção `FormatException` que é gerada pelo método `float.Parse()`. Adicionalmente, o tratamento de eventos em C# é diferente do Visual Basic não existindo em C# o equivalente à palavra reservada `Handles` do VB; para associar o evento ao *handler*, o Visual Studio adicionou a seguinte linha de código no método de inicialização da página web (`InitializeComponent`):

```
this.btnAdicao.Click += new System.EventHandler(this.btnAdicao_Click);
```

Compilar e testar.

Para colocar os restantes botões a funcionar, basta criar os métodos para os eventos de click e colocar nesses métodos o seguinte código.

```
private void btnSubtracao_Click(object sender, System.EventArgs e)
{
   IAritmetica comp = new AritmeticaImpl();

   try
   {
      lblResultado.Text = comp.Subtracao(float.Parse(txtOp1.Text),
                          float.Parse(txtOp2.Text)
                          ).ToString();
   }
   catch (FormatException ex)
   {
      lblResultado.Text = "";
   }
}

private void btnMultiplicacao_Click(object sender,
                                    System.EventArgs e)
{
   IAritmetica comp = new AritmeticaImpl();

   try
   {
      lblResultado.Text = comp.Multiplicacao(float.Parse(txtOp1.Text),
                             float.Parse(txtOp2.Text)
                             ).ToString();
   }
   catch (FormatException ex)
   {
      lblResultado.Text = "";
   }
}
```

```
private void btnDivisao_Click(object sender, System.EventArgs e)
{
   IAritmetica comp = new AritmeticaImpl();

   try
   {
      lblResultado.Text = comp.Divisao(float.Parse(txtOp1.Text),
                           float.Parse(txtOp2.Text)
                           ).ToString();
   }
   catch (FormatException ex)
   {
      lblResultado.Text = "";
   }
   catch (ApplicationException ex)
   {
      lblResultado.Text = "Não é possivel dividir por zero";
   }
}
```

Compilar e testar.

Como conclusão podemos dizer que criar componentes em .net é tão fácil como criar uma classe e coloca-la numa *Class Library* para que possa ser reutilizada em formato binário. Esse componente pode então ser utilizado de forma igual em vários tipos de clientes e em várias linguagens de programação. Para se conseguir reutilizar o componente apenas necessitamos de adicionar uma referência ao componente/projecto.

4.7 Melhorias propostas e questões

1 Criar um cliente em Winforms usando linguagem C# e com botões separados para cada operação

2 Criar um novo serviço no componente para concatenação de duas string e modificar os clientes anteriores para testar este novo método.

3 Pesquise o significado da palavra reservada `params` em C# e implemente o seguinte método:

```
public string Concatenar(params string[] args)
```

implemente num dos clientes uma função para teste deste novo método.

5 Guião de trabalho: Evolução do componente para operações aritméticas

5.1 introdução

Um aspecto importante do desenvolvimento orientado aos componentes é a evolução das interfaces dos mesmos. No caso do componente de aritmética desenvolvido na secção 4.3, vamos supor que pretendemos lançar para o mercado uma nova versão do componente com funcionalidades adicionais, por exemplo, a operação potência.

Um aspecto muito importante a ter em conta, é que uma vez uma interface publicada (ou seja, potencialmente utilizada por outras aplicações), essa interface torna-se **imutável**. Isto é, **não se devem alterar as interfaces após a sua definição e publicação** por forma a manter retro-compatibilidade. Caso se pretenda aumentar funcionalidades cria-se uma nova interface[11].

5.2 Passos preparatórios

NOTA: *este passo apenas é necessário para efeitos desta demonstração*

Antes de começar as alterações vamos copiar o directório "bin" da aplicação de teste de consola para um directório temporário para mais tarde demonstrar que a evolução do componente não implicou alterações nem recompilação das aplicações clientes já existentes.

5.3 Alterações ao componente

Vamos abrir o projecto anterior e criar uma nova interface chamada `IAritmetica2` que deriva de `IAritmetica` de acordo com o seguinte código:

[11] Na plataforma .net em concreto não seria necessário criar uma interface separada já que existe mecanismos de versionamento de componentes que facilitam a evolução dos mesmos. No entanto de um ponto de vista conceptual a interface é o contrato estabelecido entre o cliente e o prestador de serviços (componente), sendo por isso mais explícito se todos os acessos ao serviço forem efectuados através da interface específica. Para saber mais informação sobre os mecanismos de controlo de versões em .net consultar http://msdn.microsoft.com/library/en-us/cpguide/html/cpconassemblyversioning.asp.

```
public interface IAritmetica2 : IAritmetica
{
   double Potencia(float b, int expoente);
   double RaizQuadrada(float n);
}
```

Vamos agora implementar esta nova interface. Para isso vamos abrir o ficheiro da classe `AritmeticaImpl`. No código vamos acrescentar a declaração da interface que pretendemos implementar (`IAritmetica2`). De reparar que o Visual Studio irá perguntar se pretendemos gerar automaticamente o esqueleto para os métodos definidos na interface, bastando para tal pressionar na tecla TAB (Figura 41).

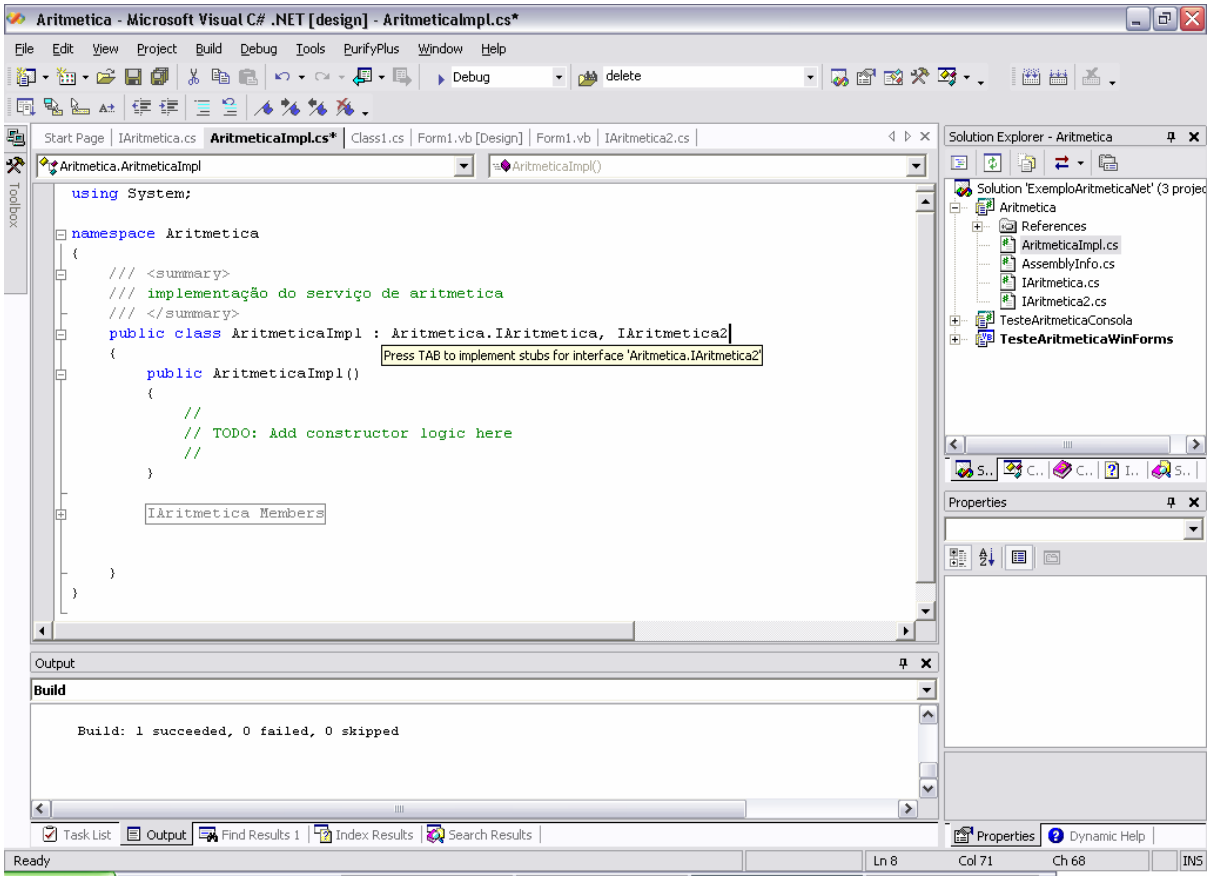

Figura 41 – geração automática do esqueleto da implementação de uma interface

Para implementar os métodos iremos utilizar os métodos existentes na classe `Math`.

```
public double Potencia(float b, int expoente)
{
   return Math.Pow(b, expoente);
}

public double RaizQuadrada(float n)
{
   return Math.Sqrt(n);
}
```

Compilar, não deve dar erros.

5.4 Aplicação de teste

Para experimentar esta nova versão do componente iremos criar uma nova aplicação de teste adicionando um novo projecto de consola em C#, denominado “TesteArti2” à solução.

Adicionar a referência ao componente.

Para testar colocar o seguinte código:

```
// criar instância do componente e requisitar nova interface
Aritmetica.IAritmetica2 comp = new Aritmetica.AritmeticaImpl();

int b = 2;
int e = 2;
float n = 25;

System.Console.WriteLine("{0}^{1} = {2}\n", b, e,
                     comp.Potencia(b, e)
                     );
System.Console.WriteLine("raiz quadrada de {0} = {1}\n", n,
                     comp.RaizQuadrada(n)
                     );
```

Definir este novo projecto como *startup project*.

Compilar (não deve dar erros) e testar.

5.5 Teste da aplicação cliente para versão anterior

Efectuar os seguintes passos:

1. abrir uma janela do explorer ou linha de comando no directório temporário para onde se copiou a aplicação de teste
2. executar o programa
3. verificar na janela do explorer a versão da DLL correspondente ao nosso componente (Figura 42)

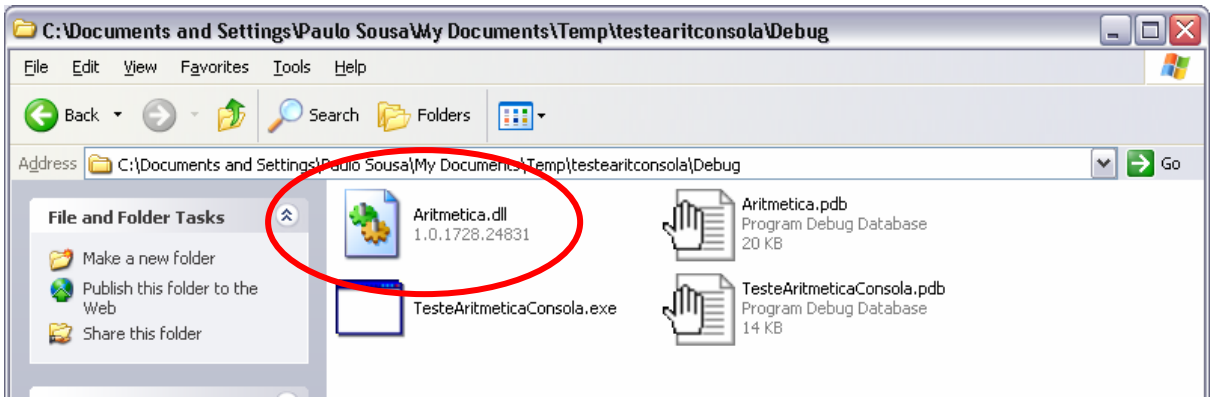

Figura 42 – versão da DLL da 1ª versão do componente

4. abrir uma nova janela do explorer no directório "bin" correspondente ao componente e verificar a versão da DLL (Figura 43).[12]

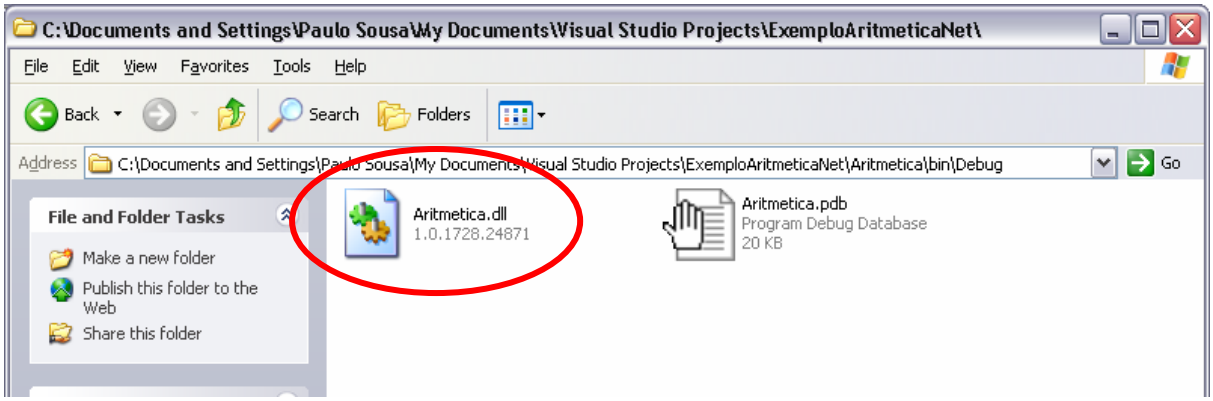

Figura 43 – versão da DLL para a 2ª versão do componente

5. copiar a nova DLL para o directório temporário
6. executar a aplicação e verificar que continua a funcionar.

Com estes passos consegue-se demonstrar que se pode evoluir a aplicação e que como se respeitou o **princípio da imutabilidade das interfaces** as aplicações clientes já existentes continuam a funcionar sem necessidade de alteração de código nem sequer recompilação .

[12] As versões das DLL existentes no vosso computador serão diferentes das apresentadas na figura. O que importa reter é que a versão existente é diferente da nova versão compilada após as alterações.

5.6 Questões

1 Tendo em conta as alterações pedidas para efectuar nos pontos 2 e 3 da secção 4.7 Melhorias propostas e questões (pág. 53):

   a. Verifique se implementou uma nova interface ou alterou a interface existente já publicada. Reflicta sobre a decisão que tomou.

   b. será que esse serviço devia ser implementado neste componente ou num componente separado?

2 Suponha que deseja adicionar um serviço para aritmética com números complexos (x+iy). Como procederia para adicionar esse serviço ao componente? Das quatro hipóteses seguintes indique vantagens e desvantagens de cada uma e escolha justificadamente a opção que seguiria:

   a. Alterava a interface existente

   b. Aproveitava a alteração actual e incluía esses métodos na interface `IAritmetica2`

   c. Criava evolução da interface existente (eg., `IAritmetica3`)

   d. Criava nova interface, `IArtimeticaComplexa`, separada

6 Guião de trabalho: Separação da criação de classes

6.1 Introdução

Um dos problemas do componente que temos vindo a trabalhar é que continua a expor alguns pormenores de implementação internos às aplicações clientes, nomeadamente a classe que implementa os serviços. Essa situação nem sempre é desejável.

Por exemplo, a equipa de desenvolvimento podia querer separar a implementação da 2ª versão da interface da 1ª e para isso criar uma nova classe (ex, `Aritmetica2Impl`). No entanto, essa situação traria alguma confusão aos utilizadores do componente pois agora teriam duas interfaces e duas classes para perceber e necessitariam de saber que apenas poderiam usar a interface `IAritmetica` com a classe `ArtimeticaImpl` e a interface `IAritmetica2` com a classe `Aritmetica2Impl`.

Por outro lado, a solução apresentada também coloca alguns problemas, pois caso se pretenda estender os serviços com novas interfaces, a classe de implementação começará a ficar muito grande e aumentará a dificuldade de manutenção de código da mesma.

Uma solução possível é recorrer ao padrão[13] Factory e delegar a criação das classes de implementação de serviços numa classe utilitária – a fábrica.

6.2 Alteração do componente

Vamos abrir o projecto do componente e criar uma nova classe chamada `Factory` com os seguintes métodos:

```
public class Factory
{
   public static IAritmetica CreateIAritmeticaService()
   {
      return new AritmeticaImpl();
   }

   public static IAritmetica2 CreateIAritmetica2Service()
   {
      return new AritmeticaImpl();
   }
}
```

Esta classe passa a ser então o ponto de contacto com a aplicação cliente. O utilizador do componente apenas necessita saber as interfaces existentes e requisitar à fábrica um objecto que implemente o serviço desejado. De notar que se optou por uma regra de nomenclatura `CreateXXXService`, em que `XXX` representa a interface desejada.

Futuras evoluções apenas necessitarão da criação de novos métodos fábrica. Deve-se ter em atenção que o princípio de imutabilidade das interfaces também deve aqui ser respeitado, não sendo aconselhável alterar os métodos fábrica já existentes, mas sim criar novos métodos.

Como agora a criação das classes está separada numa classe utilitária, essas classes podem evoluir de forma independente. Um outro passo necessário é "esconder" as classes de implementação do exterior, para isso vamos alterar o tipo de visibilidade da classe `AritmeticaImpl` de `public` para `internal`. O que faz com que esta classe não possa ser instanciada fora do *assembly* (neste caso, a DLL) onde é definida.

[13] Um padrão é uma solução tipificada para um problema recorrente. Podem encontrar mais informação em "Design patterns : elements of reusable object-oriented software", Erich Gamma, Richard Helm, Ralph Johnson e John Vissides, e obter implementações base em C# em http://www.dofactory.com/Patterns/Patterns.aspx .

```
internal class AritmeticaImpl : IAritmetica , IAritmetica2
{
    ...
}
```

6.3 Aplicação de teste

Para experimentar esta nova versão do componente iremos criar uma nova aplicação de teste adicionando um novo projecto de consola em C#, denominado “TesteFactory” à solução.

Adicionar a referência ao componente.

Para testar colocar o seguinte código:

```
int op1 = 5;
int op2 = 7;

// obter implementação de serviço desejado
Aritmetica.IAritmetica comp =
              Aritmetica.Factory.CreateIAritmeticaService();

//invocar o serviço pretendido
Console.WriteLine("Adição({0}, {1}) => {2}",
                    op1, op2, comp.Adicao(op1, op2)
                    );
```

Compilar (não deve dar erros) e testar.

6.4 Implicações nas aplicações já existentes

Estas alterações têm um grande impacto nas aplicações já existentes implicando a alteração do código fonte e recompilação para utilização da classe fábrica.

Uma hipótese de minimizar esse impacto é deixar a classe `AritmeticaImpl` com visibilidade `public`. Desse modo todas as aplicações existentes continuam a funcionar criando directamente a classe de implementação, enquanto que os novos clientes usariam apenas a classe de fábrica. Nesse caso, todas as implementações de novos serviços deveriam ser colocadas em novas classes com visibilidade `internal`.

7 Guião de trabalho: Carregamento dinâmico de componentes

7.1 Introdução

Até ao momento vimos como criar e usar componentes numa aplicação, mas em situações onde a solução do Visual Studio contém o projecto do componente e da aplicação cliente. Desta forma ao adicionarmos a referência do projecto do componente ao projecto do da

aplicação cliente estamos a definir uma ligação estática entre os dois projectos (estática no sentido que as ligações e dependências são conhecidas em tempo de compilação). No entanto, nem sempre podemos ou queremos fazer as coisas desta forma, porque o componente que queremos usar foi desenvolvido por terceiros e não temos acesso ao projecto para adicionar à solução, ou porque queremos escolher o componente apenas em tempo de execução da aplicação e não na compilação.

7.2 Ligação estática a componentes de terceiros (sem projecto no Visual Studio)

Em situações em que não temos acesso ao projecto do componente que queremos usar, podemos estabelecer uma ligação estática (em tempo de compilação) entre a aplicação cliente (ou componente cliente) e a DLL do componente que queremos usar.

Suponha que queremos usar um componente desenvolvido por outra equipa do qual apenas recebemos a DLL e instalamos na nossa máquina. Basta no nosso projecto adicionar uma referência (`Project` → `Add Reference`) e em seguida escolher o botão Browse (Figura 44).

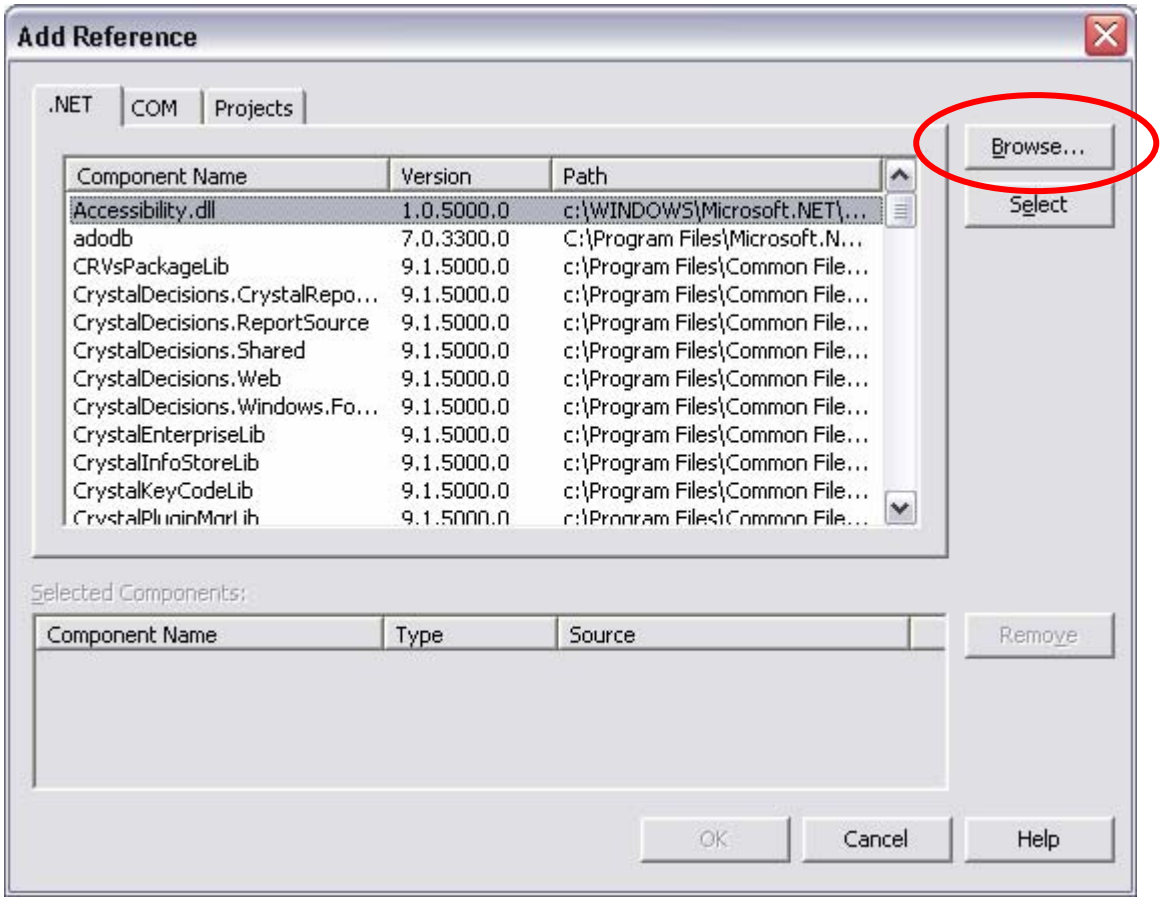

Figura 44 – Adicionar referência a componente de terceiros

Após indicarmos o caminho para a DLL do componente que queremos usar, tudo o resto é igual. De notar que a opção de adicionar uma referência a um projecto existente na solução está na realidade a criar uma referência deste tipo.

7.3 Carregamento dinâmico de componentes

Para carregar dinamicamente um componente (em tempo de execução) vamos utilizar a API de *Reflection*[14] do .net. Esta API permite entre outras coisas inspeccionar uma classe para obter informação sobre os seus membros (métodos, propriedades e campos) e invocar dinamicamente esses membros.

Em primeiro lugar é conveniente fazer referência no código ao *namespace*:

```
using System.Reflection;
```

Para carregar dinamicamente uma DLL usamos o método de classe `LoadFrom` da classe `Assembly`:

```
Assembly asb = Assembly.LoadFrom(componentFileName);
```

por exemplo:

```
Assembly asb = Assembly.LoadFrom(@“c:\temp\meucomp.dll”);
```

Este método lança uma excepção do tipo `System.IO.FileNotFoundException` caso não encontre a DLL.

Para criar uma nova instância de uma classe definida neste *assembly* e que possua um construtor sem parâmetros podemos usar o método `CreateInstance`:

```
meuTipo obj = (meuTipo)asb.CreateInstance(nome_completo_classe);
```

por exemplo:

```
IMyInterface obj =
       (IMyInterface)asb.CreateInstance(“MyNameSpace.MyClass”);
```

[14] A API de *Reflection* é muito extensa e relativamente complexa e um estudo sobre essa API sai fora do âmbito deste documento, mas podem consultar mais informação em http://msdn.microsoft.com/library/en-us/cpguide/html/cpconReflectionOverview.asp e http://msdn.microsoft.com/library//en-us/cpref/html/frlrfsystemreflection.asp

Também é possível obter directamente meta-informação sobre um tipo específico e em seguida usar essa meta-informação para criar instâncias ou invocar métodos. Por exemplo, supondo a existência de um tipo chamado `MinhaClasse` no *namespace* `MeuNamespace` que possui um construtor sem parâmetros e tem um método público chamado `MeuMetodo` que aceita um parâmetro do tipo `int`. Poderíamos usar o seguinte extracto de código para criar uma nova instância desse tipo e invocar o método com o argumento 123:

```
// obter meta tipo
Type typ = asb.GetType("MeuNamespace.MinhaClasse");

// obter constructor sem parâmetros
ConstructorInfo ctor = typ.GetConstructor(Type.EmptyTypes);

// criar nova instância por reflexão
object obj = ctor.Invoke(null);

// obter método desejado
MethodInfo mi = typ.GetMethod("MeuMetodo", new Type[]{typeof(int)} );

// invocar o método com os argumentos desejados na instância criada
mi.Invoke(obj, new object[]{123} );
```

7.4 Um exemplo

Vamos supor uma situação em que estamos a desenvolver uma aplicação que necessita de transferir dados para sistemas externos já existentes nos nossos clientes. No entanto, os nossos clientes actuais possuem diferentes sistemas que necessitam de informação de maneira diferente, e adicionalmente é possível que novos clientes possuam outros sistemas que desconhecemos de momento.

A solução pretendida tem que ser flexível para suportar os clientes existentes bem como futuros e não queremos estar a recompilar a aplicação de cada vez que necessitamos de ligar com um novo sistema do cliente. Vamos então aplicar um modelo baseado em *providers* para solucionar este problema (Figura 45).

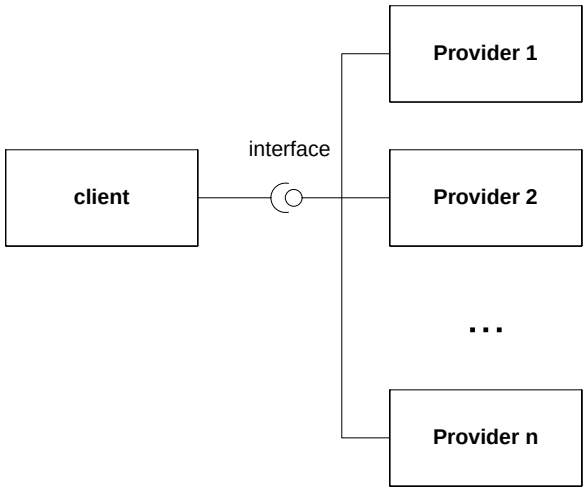

Figura 45 – modelo de providers

O "truque" neste modelo é definir uma interface que o cliente (aplicação ou componente) necessitam e que os fornecedores implementam, sendo possível haver várias implementações (*providers*) para essa mesma interface.

Para carregar dinamicamente um desses fornecedores vamos usar o código apresentado na secção anterior. Agora falta definir como é que o cliente sabe qual o fornecedor a carregar, para isso vamos usar um ficheiro de configuração.

7.4.1 Passo 1: criar solução e projectos iniciais

Criar uma solução vazia denominada ExemploCarregamentoDinamico. Nessa solução criar um projecto windows que será a aplicação cliente e chamar-lhe DemoApp. Criar um outro projecto do tipo *class library* e chamar-lhe ImportExport.

7.4.2 Passo 2: definir interface dos *providers*

No projecto ImportExport definir uma interface pública semelhante à seguinte (não esquecer o `using System.Collections`):

```
public interface IImportExport
{
   void Export(IList data);
   IList Import();
}
```

O que vamos importar ou exportar são colecções de informação sobre pessoas e para tal necessitamos também definir essa estrutura de dados. Tendo em conta que esta classe funciona como uma estrutura de dados e não como uma classe de objectos, os campos da classe poderiam ser públicos em vez de privados, evitando dessa forma o uso de propriedades do C#. No entanto as características de *data binding* dos controlos gráficos do

Windows Forms apenas trabalham com propriedades públicas pelo que se optou por definir as propriedades para possibilitar o uso de obejctos desta classe em elementos tipo DataGrid.

```
public class PessoaInfo
{
   public PessoaInfo() { }

   public PessoaInfo(string n, int i)
   {
      _nome = n;
      _idade = i;
   }

   public string Nome
   {
      get { return _nome; }
      set { _nome = value; }
   }

   public int Idade
   {
      get { return _idade; }
      set { _idade = value; }
   }

   public override string ToString()
   {
      return _nome + "(" + _idade + ")";
   }

   private string _nome;
   private int _idade;
}
```

Compilar.

7.4.3 Passo 3: implementar aplicação cliente

Desenhar a interface gráfica da aplicação DemoApp de acordo com a figura seguinte.

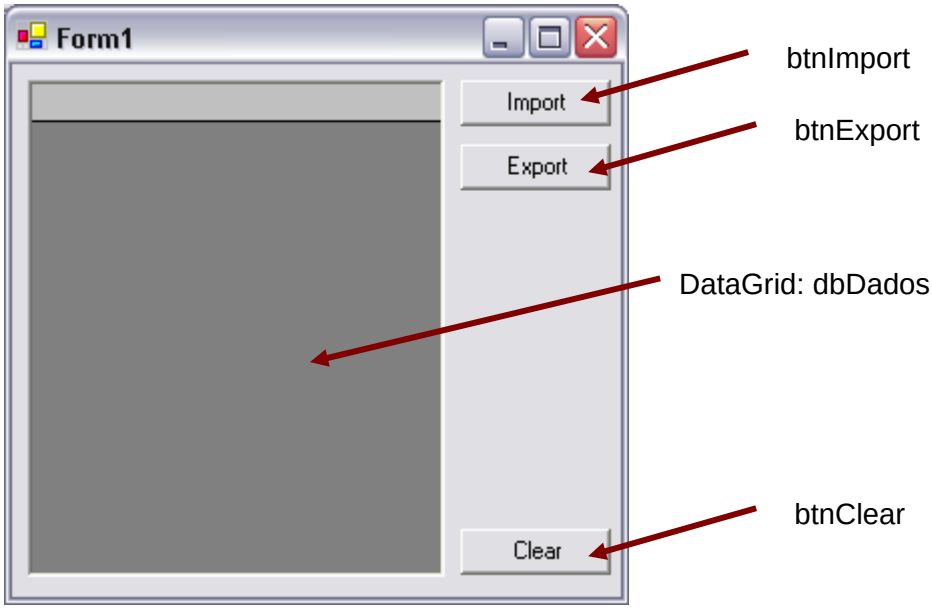

Figura 46 - GUI da aplicação demo

Temos que implementar o processo de carregamento dinâmico do componente e respectiva criação do objecto que implementa a interface `IImportExport` para que a aplicação funcione. Como vimos atrás no capítulo 6, devemos esconder os pormenores de criação das classes em classes fábrica. Esta situação é ideal para uma fábrica já que o processo de criação é mais complexo que um simples new. Para tal vamos criar neste projecto (DemoApp) uma classe chamada `ImportExportFactory` com o seguinte código (mais tarde implementaremos a carregamento dinâmico, por agora apenas pretendemos criar o esqueleto da aplicação e garantir a compilação).

```
using ImportExport;
...

public class ImportExportFactory
{
   public ImportExportFactory()
   {
   }

   public IImportExport CreateProvider()
   {
      return null;
   }
}
```

Para que possamos usar o tipo `IImportExport` temos que adicionar uma referência ao projecto que define a interface.

Adicionar o seguinte código no *handler* dos botões correspondentes:

```
private void btnClear_Click(object sender, System.EventArgs e)
{
   dgDados.DataSource = null;
}

private void btnImport_Click(object sender, System.EventArgs e)
{
   ImportExportFactory fact = new ImportExportFactory();
   ImportExport.IImportExport prov = fact.CreateProvider();
   dgDados.DataSource = prov.Import();
}

private void btnExport_Click(object sender, System.EventArgs e)
{
   ImportExportFactory fact = new ImportExportFactory();
   ImportExport.IImportExport prov = fact.CreateProvider();
   prov.Export( (IList)dgDados.DataSource );
}
```

Obviamente, poderíamos usar uma variável da classe `Form1` para declarar o *provider* e não criar uma variável local de cada vez que necessitamos de uma operação de importação ou exportação (essa alteração fica a cargo de cada leitor).

Compilar (não deve dar erros).

7.4.4 Passo 4: adicionar ficheiro de configuração

Vamos então adicionar um ficheiro de configuração à nossa aplicação onde colocaremos a informação necessária para indicar qual o *provider* a usar para importação e exportação de dados. Posteriormente, em cada instalação nos clientes este ficheiro seria alterado por forma a ter a configuração indicada para cada caso.

Escolher a opção `Project` → `Add New Item`. Na janela que aparece escolher o item `Application configuration file` do grupo `Local Project Itens` (Figura 47)

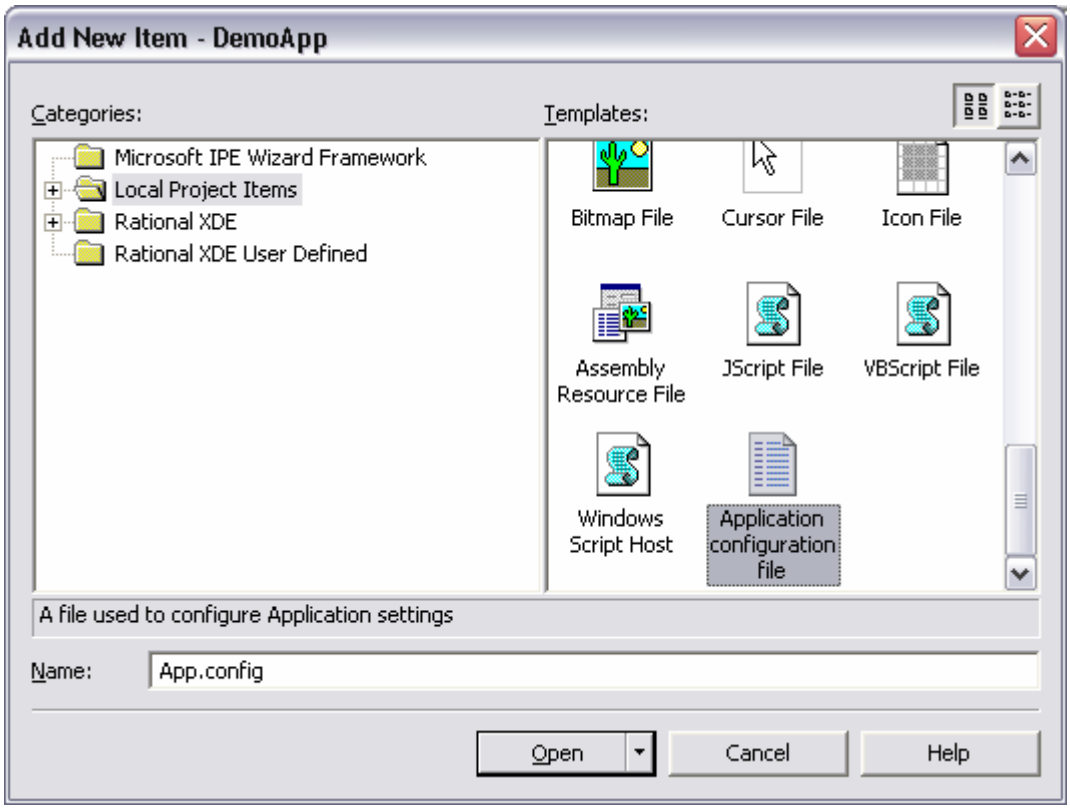

Figura 47 – janela add new item

Não alterar o nome do ficheiro e pressionar o botão `Open`.

Alterar o conteúdo do ficheiro criado de acordo com as linhas seguintes indicadas a *bold* (ter atenção a maiúsculas e minúsculas).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="ImpExpProv" value=""/>
  </appSettings>
</configuration>
```

Mais tarde iremos atribuir um valor a esta chave no ficheiro de configuração e será esse valor que a fábrica vai ler para carregar o *provider* correcto.

Para ler valores do ficheiro de configuração a *framework* .Net fornece a classe `System.Configuration.ConfigurationSettings` cujo membro `AppSettings` fornece acesso a cada um dos pares chave/valor definidos na secção `<appSettings>` do ficheiro de configuração.

Para que o ficheiro de configuração seja reconhecido pela aplicação, este deve ter o mesmo nome da aplicação com a extensão .config (ex., MeuProg1.exe.config). Ao usar o Visual

Studio, o próprio IDE encarrega-se de renomear e copiar o ficheiro app.config que acabamos de criar para o directório correcto.

7.4.5 Passo 5: implementar *provider* de teste

Vamos temporariamente parar o desenvolvimento da aplicação cliente e construir um *provider* básico que possamos usar para teste da aplicação. Vamos para isso criar um novo projecto do tipo *class library* chamado DummyProvider.

Adicionar uma referência para o projecto contendo a interface de importação/exportação.

Alterar a classe `Class1` para o seguinte código:

```
using System.IO;
using System.Collections;
using ImportExport;

public class Class1 : IImportExport
{
   public Class1() { }

   public void Export(IList data)
   {
      TextWriter wr = new StreamWriter(@"c:\temp\dummy.txt");

      foreach (PessoaInfo pi in data)
      {
         wr.WriteLine(pi);
      }
      wr.Close();
   }

   public IList Import()
   {
      IList ret = new ArrayList();

      ret.Add(new PessoaInfo("joao", 29));
      ret.Add(new PessoaInfo("carla", 25));
      ret.Add(new PessoaInfo("margarida", 32));

      return ret;
   }
}
```

Compilar (não deve dar erros).

7.4.6 Passo 6: implementar fábrica

Vamos agora implementar a nossa fábrica colocando o código responsável por ler o ficheiro de configuração, carregar a DLL com o componente indicado para memória e com base nos tipos definidos nesse componente criar uma nova instância do tipo desejado (neste caso uma classe que implemente a interface `IImportExport`). Assim no projecto DemoApp

colocar o seguinte código no método `CreateProvider` da classe `ImportExportFactory`:

```
public ImportExport.IImportExport CreateProvider()
{
  // ler ficheiro de configuração
  string providerName =
System.Configuration.ConfigurationSettings.AppSettings["ImpExpProv"];

  if (providerName == null || providerName.Length == 0)
    throw new ApplicationException("Falta configurar o provider de
importação/exportação");

  try
  {
    // carregar assembly desejado
    Assembly asb = Assembly.LoadFrom(providerName);

    // pesquisar classes que implementam a interface desejada
    foreach (Type typ in asb.GetExportedTypes())
    {
      // se este tipo implementa a interface desejada
      if (typeof(IImportExport).IsAssignableFrom(typ))
      {
        // criar objecto fornecedor de serviço

        // 1. obter construtor sem parâmetros
        ConstructorInfo ctor = typ.GetConstructor(Type.EmptyTypes);
        // 2. requisitar nova instância deste tipo
        object obj = ctor.Invoke(null);

        return (ImportExport.IImportExport)obj;
      }
    }
  }
  catch (System.IO.FileNotFoundException ex)
  {
    throw new ApplicationException("provider não encontrado: " +
providerName);
  }

  // não encontrou nenhum tipo que implementasse a interface desejada
  return null;
}
```

Para que a aplicação funcione é necessário indicar no ficheiro de configuração o *provider* a usar (indicar o caminho para a DLL):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="ImpExpProv" value="c:\temp\DummyProvider.dll"/>
  </appSettings>
</configuration>
```

Compilar, e testar (sugestão: fazer *debug* no método `CreateProvider` da classe `ImportExportFactory`). Após testarem a exportação não se esqueçam de ir verificar o ficheiro criado. Experimentem editar o conteúdo de uma das células na grelha do *form* e em seguida exportar.

NOTA: *o código do método* `CreateProvider` *da classe* `ImportExportFactory` *poderia ser mais simples se colocássemos directamente no ficheiro de configuração o nome da classe que implementa o serviço evitando dessa forma o ciclo à procura de uma classe que implemente a interface desejada. Isto poderia ser necessário se quiséssemos ter numa única DLL vários providers.*

7.4.7 Passo 7: implementar *provider* CSV

Vamos agora criar outro *provider* (o que seria um *provider* real o nosso cenário a ser usado para ligar a aplicações do cliente). Neste caso vamos simular um sistema externo que consome e produz ficheiros CSV (*comma separated values*) no formato:

```
Joao, 29
Margarida, 32
```

Vamos começar por criar um projecto (pode ser na mesma solução ou noutra) do tipo *class library* e chamar-lhe CsvProvider. Este novo projecto vai necessitar de referenciar o componente onde está definida a interface `IImportExport` bem como a classe `PessoaInfo`, por isso não esquecer de adicionar a referência.

No ficheiro que define a classe `Class1` vamos renomear a classe para `CsvProviderImpl`. Para implementar esta funcionalidade vamos ler e gravar ficheiros em disco “à moda antiga”. Colocar na declaração da classe que esta implementa a interface ImportExport e implementar os respectivos métodos.

```
using System.Collections;
using System.IO;
using ImportExport;

public class CsvProviderImpl : IImportExport
{
   public CsvProviderImpl() { }

   public void Export(IList data)
   {
      TextWriter wr = new StreamWriter(@"c:\temp\CsvData.csv");
      if (data != null)
      {
         foreach (PessoaInfo pi in data)
         {
            wr.WriteLine("{0}, {1}", pi.Nome, pi.Idade);
         }
      }
```

```
      wr.Close();
   }

   public IList Import()
   {
      IList ret = new ArrayList();

      TextReader rd = new StreamReader(@"c:\temp\CsvData.csv");
      string line;
      while ((line = rd.ReadLine()) != null)
      {
         int idx = line.LastIndexOf(',');
         string nome = line.Substring(0, idx).Trim();
         string idade = line.Substring(idx+1);
         PessoaInfo pi = new PessoaInfo(nome, int.Parse(idade));
         ret.Add(pi);
      }
      rd.Close();

      return ret;
   }
}
```

Antes de testar, criar um ficheiro de texto chamado CsvData.csv na directoria c:\temp e colocar um conteúdo do género:

```
joao, 30
carla, 26
margarida, 32
```

Compilar, alterar o ficheiro de configuração da aplicação para usar este novo componente e testar.

7.4.8 Passo 8: implementar *provider* XML

Vamos agora criar outro *provider*, neste caso para simular um sistema externo que consome e produz ficheiros XML no formato:

```
<pessoas>
   <pessoa nome="joao" idade="29" />
</pessoas>
```

Vamos criar um projecto (na mesma solução ou noutra) do tipo *class library* e chamar-lhe XmlProvider. Este novo projecto vai necessitar de referenciar o componente onde está definida a interface `IImportExport` bem como a classe `PessoaInfo`, por isso não esquecer de adicionar a referência.

No ficheiro que define a classe `Class1` vamos renomear a classe para `XmlProviderImpl`. Em seguida vamos usar o suporte de XML no .net para criar um documento XML com os dados pretendidos e gravar para disco (no caso da exportação) ou

carregar um ficheiro de disco para um documento XML e construir uma lista que possa ser devolvida à aplicação no caso da importação.

```
using System.Collections;
using System.Xml;
using ImportExport;

public class XmlProviderImpl : IImportExport
{
  public XmlProviderImpl() { }

  public void Export(IList data)
  {
    // criar documento XML
    //(documento é construido totalmente em memória)
    XmlDocument doc = new XmlDocument();
    XmlElement root = doc.CreateElement("pessoas");
    if (data != null)
    {
      foreach (PessoaInfo pi in data)
      {
        XmlElement no = doc.CreateElement("pessoa");
        no.SetAttribute("nome", pi.Nome);
        no.SetAttribute("idade", pi.Idade.ToString());
        root.AppendChild(no);
      }
    }
    doc.AppendChild(root);

    // gravar documento em disco
    doc.Save(@"c:\temp\XmlData.xml");
  }

  public IList Import()
  {
    // criar lista com base nos elementos existentes
    IList ret = new ArrayList();

    // ler documento do disco em modo streaming
    // (optimiza utilização de memoria)
    XmlReader reader = new XmlTextReader(@"c:\temp\XmlData.xml");
    while (reader.Read())
    {
      if (reader.NodeType == XmlNodeType.Element &&
          reader.Name == "pessoa")
      {
        string nome = reader.GetAttribute("nome");
        string idade = reader.GetAttribute("idade");

        PessoaInfo  pi = new PessoaInfo(nome,int.Parse(idade));
        ret.Add(pi);
      }
    }
    reader.Close();
    return ret;
  }
}
```

Antes de testar, criar um ficheiro de texto chamado XmlData.xml na directoria c:\temp e colocar um conteúdo do género:

```
<pessoas>
   <pessoa nome="joao" idade="30" />
   <pessoa nome="maria" idade="26" />
   <pessoa nome="margarida" idade="32" />
</pessoas>
```

Compilar, alterar o ficheiro de configuração da aplicação para usar este novo componente e testar.

8 Conclusões

O importante compreender após a conclusão deste guião não é a utilizar o Visual Studio, nem a linguagem C# nem o .NET. O importante são três conceitos:

- Interface
- Componente
- Cliente

A **interface** define o contrato do serviço a ser prestado; aquilo que interessa às partes envolvidas nesta interacção. Sendo um contrato **a interface é imutável**, e caso seja necessário evoluir ou alterar o contrato deve-se criar um novo contrato (i.e., interface). Um ou mais **componentes** irão depois *implementar* essa interface, podendo cada um deles implementar mais que uma interfaces. Por seu lado, um ou mais **clientes** (aplicações executáveis ou outros componentes) irão *consumir* este serviço. Na maioria dos casos, um cliente não necessita de conhecer especificamente o fornecedor de serviços; tanto lhe faz que seja A ou B desde que o contrato seja cumprido. De igual forma, um componente não necessita, *nem deve*, fazer assunções sobre quem é o seu cliente.

Anexo 1 Utilização do .Net SDK em vez do Visual Studio

1.1 Introdução

Embora o Visual Studio seja uma ferramenta poderosa e de grande produtividade, não é obrigatória para o desenvolvimento de aplicações na plataforma .net. todo o desenvolvimento pode ser feito usando um simples editor de texto (e.g., Notepad, EditPlus) e os compiladores na linha de comando. As ferramentas de desenvolvimento estão disponíveis de forma grátis no site da Microsoft e tomam o nome de .Net Framework SDK. Podem descarregar o SDK a partir do URL http://msdn.microsoft.com/netframework/downloads/updates/default.aspx (caso tenham o Visual Studio instalado, o SDK já se encontra instalado na máquina na sub-directoria SDK debaixo do directório de instalação do VS). Necessitam de instalar o *runtime* do .net antes de instalar o SDK.

Após instalarem o SDK, podem consultar em bin\StartTool.htm uma página com documentação das várias ferramentas disponíveis. Os componentes de *runtime* são instalados no directório do windows debaixo do subdirectório Microsoft.NET

1.2 Preparar o ambiente de desenvolvimento

Após instalar o SDK existirá um grupo de programas no windows com os *shortcuts* para algumas ferramentas e documentação. Um dos *shortcuts* mais importantes é um *command file* para criar uma consola do sistema operativo com o *path* e variáveis de ambiente devidamente definidos. Essa opção de menu deve ser algo do género ".net sdk command prompt" ou "set build environment". Também podem procurar no disco por um ficheiro tipo sdkvars.bat, vcvars32.bat ou setenv.cmd

1.3 Compilador C# - csc

Opções mais comuns:

- /help – obter ajuda
- /out ou /o– especificar nome do ficheiro de output a criar
- /target ou /t– especificar tipo de output a criar (executável de consola, executável de janela ou *class library*)
- /reference ou /r – especificar bibliotecas adicionais a referenciar na compilação

Exemplos:

- `csc MinhaClasse.cs`
- `csc class1.cs class2.cs /out:Programa /target:exe`
- `csc App.cs Proc.cs /out:WinPrograma /target:winexe`
- `csc IXpto.cs XptoImpl.cs XptoUtil.cs Teste.cs IXpto2.cs /out:Componente /target:library /r:c:\lib3party\CryptoTools.dll /r:c:\lib3party\CommHelper.dll`

1.4 NMake

Este utilitário permite gerir o processo de *build* das aplicações, permitindo definir regras de dependências para compilação dos diferentes ficheiros da aplicação. Existem três conceitos fundamentais no NMake:

- *Targets* – especifica algo a construir (um executável ou *class library*) ou um pseudo-*target* para construir regras mais elaboradas. Um pseudo-*target* permite definir dependências entre *targets*.
- *Dependents* – especifica os ficheiros de que um *target* necessita para ser construído/compilado. O NMake fará uma verificação da data de última alteração de cada dependente e caso seja mais recente que o *target* procederá oa processo de *build*. Caso contrário não desperdiçará tempo na compilação desse *target*.
- *Commands* – especifica o comando a executar para construir o *target*.

Basicamente um ficheiro *makefile* contém regras do seguinte tipo:

```
target: dependente1 dependente2 ...
     comando
```

Podem encontrar informação mais completa sobre este utilitário em http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_asug_Overview.3a_.NMAKE_Reference.asp

Exemplo de um ficheiro de *build* para a solução apresentada na secção 4 seria o seguinte (gravar o ficheiro com o nome “makefile” no directório da solução).

```
#
# makefile para exemplo Aritmetica – ADAV / ISEP
#

### special target ALL for building everything in the solution
#
All: Aritmetica.dll TesteAritmeticaConsola TesteAritmeticaWinforms
```

```
#
# o componente
#
Aritmetica.dll:Aritmetica\IAritmetica.cs Aritmetica\AritmeticaImpl.cs
Aritmetica\Factory.cs
     csc /out:$@ /target:library $**

#
# a aplicação de consola
#
TesteAritmeticaConsola: Aritmetica.dll TesteAritmeticaConsola.exe

TesteAritmeticaConsola.exe: TesteAritmeticaConsola\Class1.cs
     csc /out:$@ /target:exe $** /reference:Aritmetica.dll

#
# a aplicação winforms
#
TesteAritmeticaWinforms: Aritmetica.dll TesteAritmeticaWinforms.exe

TesteAritmeticaWinforms.exe: TesteAritmeticaWinforms\Form1.vb
     vbc /main:Form1 /out:$@ /target:winexe $**  /r:mscorlib.dll
/r:System.dll /r:System.Windows.Forms.dll /r:System.Drawing.dll
/r:Aritmetica.dll
```

Macros pré-definidas (entre outras):

- $@ – o *target*
- $** – todos os dependentes
- $? – todos os dependentes que tenham um *timestamp* mais recente que o *target*

NOTA: *se ao compilar o programa VB aparecer o erro BC30002 referente às classes `ApplicationException` e `FormatException`, alterem o código fonte e utilizem o nome completo das classes, isto é, `System.ApplicationException` e `System.FormatException`*

Exemplo de um ficheiro de *build* para a solução apresentada na secção 7 seria o seguinte (gravar o ficheiro com o nome “makefile” no directório da solução).

```
#
# makefile para exemplo Carregamento dinâmico – ADAV / ISEP
#

### special target ALL for building everything in the solution
#
All: ImportExport.dll DemoApp DummyProvider CsvProvider XmlProvider

#
# o componente ImportExport
#
ImportExport.dll: ImportExport\IImportExport.cs \
ImportExport\PessoaInfo.cs
     csc /out:$@ /target:library $**
```

```
#
# o provider de teste
#
DummyProvider: ImportExport.dll DummyProvider.dll

DummyProvider.dll: DummyProvider\Class1.cs
     csc /out:$@ /target:library $** /r:ImportExport.dll

#
# o provider CSV
#
CsvProvider: ImportExport.dll CsvProvider.dll

CsvProvider.dll: CsvProvider\Class1.cs
     csc /out:$@ /target:library $** /r:ImportExport.dll

#
# o provider Xml
#
XmlProvider: ImportExport.dll XmlProvider.dll

XmlProvider.dll: XmlProvider\Class1.cs
     csc /out:$@ /target:library $** /r:ImportExport.dll

#
# a aplicação winforms
#
DemoApp:: DemoApp\App.config
     copy DemoApp\App.Config $@.config

DemoApp:: ImportExport.dll DemoApp.exe

DemoApp.exe: DemoApp\Form1.cs DemoApp\ImportExportFactory.cs
     csc /out:$@ /target:winexe $** /r:ImportExport.dll
```

1.5 Outras ferramentas

1.5.1 aspnet_regiis

Configura o IIS para utilizar ASP.net. Utilizar a opção –i para instalar novamente o suporte ASP.net no IIS em caso de problemas.

1.5.2 ILDAsm

Permite visualizar o código IL de um *assembly* (Figura 48 e Figura 49).

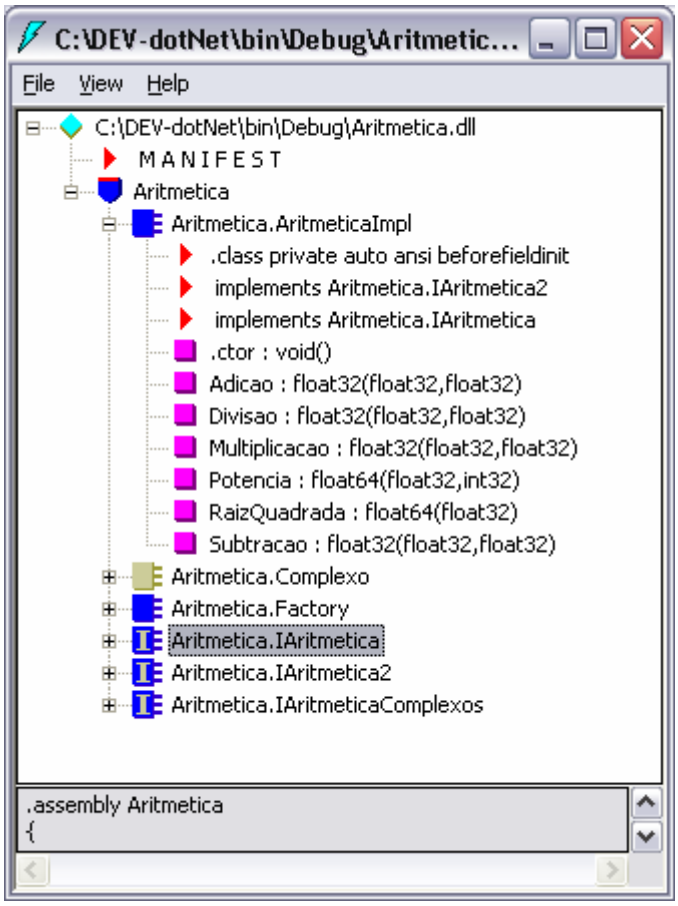

Figura 48 - ILDAsm

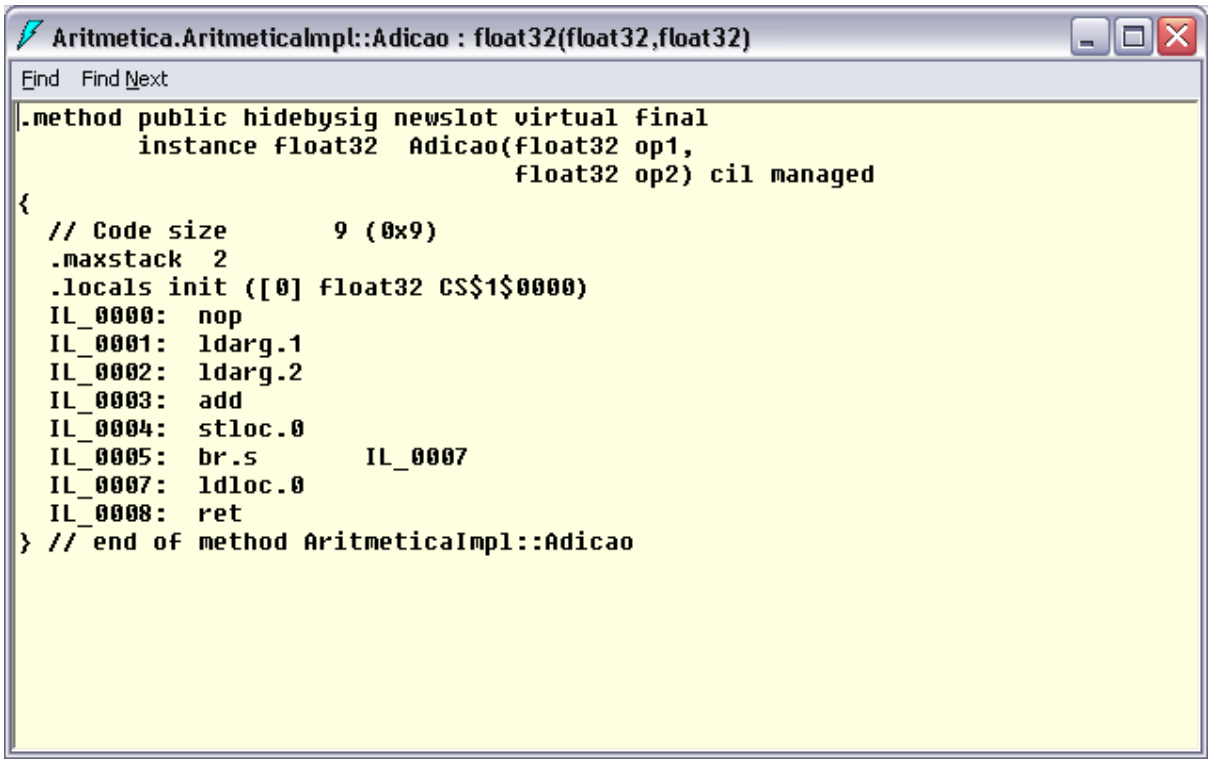

Figura 49 - Código IL de um método

1.5.3 gacutil

Permite manipular a Global Assembly Cache do .net. A GAC funciona como um repositório de *assemblies* onde podem ser colocados todos os *assemblies* a ser partilhados por várias aplicações evitando dessa forma que cada aplicação tenha a sua cópia no directório de trabalho.

Opções mais comuns:

- /l – lista o conteúdo do GAC
- /i – instala um *assembly* no GAC
- /u – remove um *assembly* do GAC

1.5.4 regasm

Regista um *assembly* .net no *registry* do windows por forma a permitir compatibilidade com clientes COM.

1.5.5 NGen

Permite gerar código nativo a partir de um *assembly* em IL na altura da instalação, evitando dessa forma que o código tenha que ser compilado pelo JIT da máquina virtual.

1.5.6 sn

Permite gerar *strong names* para um *assembly* (necessário para instalar um assembly na GAC). Para mais informações sobre *strong names* ver http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconStrong-NamedAssemblies.asp

Como utilizar:

1. Criar chave

   ```
   sn –k minhachave.snk
   ```

2. no código fonte do *assembly* a assinar colocar a seguinte linha de código (caso usem o Visual Studio, alterar no ficheiro assemblyinfo.cs ou assemblyinfo.vb)

   ```
   using System.Reflection;

   [assembly: AssemblyKeyFile("minhachave.snk")]
   ```

Anexo 2 A Linguagem C#

2.1 Introdução

- Nova linguagem tendo por base o C/C++
    - Também vai buscar inspiração ao Java ;-)
    - Mantém o investimento e *know-how* existente
- Código mais “limpo”
- Construções sintácticas especiais para tirar partido do *framework*
- Tudo são objectos
- Ficheiros com extensão .cs
- Declaração e definição de métodos no mesmo ficheiro

2.2 Tipos de dados e operadores

A tabela seguinte apresenta os diferentes tipos de dados existentes no C#.

Tabela 3 – tipos de dados existentes no C#

Categoria	Tipos existentes
Objectos	`object`
alfanuméricos	`string`, `char`
Inteiros com sinal	`sbyte`, `short`, `int`, `long`
Inteiros sem sinal	`byte`, `ushort`, `uint`, `ulong`
Virgula flutuante	`float`, `double`, `decimal`
boleanos	`bool`

- Estes tipos são *alias* para os tipos definidos na *framework*
    - Ex., `int` == `System.Int32`
- Tipos Enumerados definidos pelo utilizador

- Fortemente “tipados”
- Sem conversão automática para `int`
- Suportam operadores +, -, ++, --, &, |, ^, ~
- Pode-se definir tipo de dados base (`byte`, `short`, `int`, `long`)

```
enum DayOfWeek
{
	Monday,
	Tuesday,
	Wednesday,
	Thursday,
	Friday,
	Saturday,
	Sunday
}

enum Color : byte
{
	Red = 1,
	Green = 2,
	Blue = 4,
	Black = 0,
	White = Red | Green | Blue
}
```

A tabela seguinte apresenta os operadores existentes em C# (são os mesmo do C++). Em C# é possível redefinir operadores.

Tabela 4 – operadores existentes em C#

Categoria	Operadores
Atribuição	=
Relacionais	< <= > >= == !=
Lógicos	&& || !
Aritméticos	+ - * / % += -= *= /= ++ --
Tipos	typeof is

Os últimos operadores da tabela permitem saber o tipo de uma variável ou verificar se uma dada variável é de um determinado tipo (para mais informação sobre estes operadores consultar http://msdn.microsoft.com/library/en-us/csref/html/vclrftypeofpg.asp e http://msdn.microsoft.com/library/en-us/csref/html/vclrfispg.asp). Ex.:

```
int x = 5;
MinhaClasse c = new MinhaClasse();

if (x is int)
{
      ...
}
if (x is typeof(c))
{
      ...
}
```

2.3 Sintaxe

2.3.1 Constantes

- Pré-definidas (`null, true, false`)
- De utilizador:

```
const string Ver = “1.0b”;
```

2.3.2 Classes e namespaces

- Organização do código dentro de classes
- Classes organizadas dentro de *namespaces*

```
namespace Demo
{
      public class MyClass
      {
            ...
      }
}
```

2.3.3 Construtores

- Seguem as regras do C/C++
- Mesmo nome da classe
- Sem tipo de retorno
- Podem ter ou não argumentos

```
public class MyClass
{
      ...
      public MyClass() { ... }
      public MyClass(string title) { ... }
}
```

2.3.4 Métodos

- Sintaxe semelhante ao C/C++
- Podem ser públicos ou privados
- Suporta *overloading*

```
public class MyHelloWorld
{
      ...
      public void SayHello()
      { ... }

      private void SetTitle(String Title)
      { ... }

      private void SetTitle(String title, Color c)
      { ... }
}
```

2.3.5 Passagem de parâmetros

O C# suporta passagem de argumentos para um método, por valor e por referência. Neste último caso, é possível indicar se o parâmetro é apenas de saída - `out` - ou de entrada e saída - `ref`.

```
// passagem por valor
public void func1(int x)
{
      ...
}

// passagem por referência – apenas de saída
public void func2(out int x)
{
      ...
}

// passagem por referência – entrada e saída
public void func2(ref int x)
{
       ...
}
```

2.3.6 Herança

- Apenas existe herança simples, ou seja, uma classe apenas pode derivar de outra.

```
public class MyClassBase
{
      ...
      public void Func()
      { ... }
}
```

```
public class MyClassDeriv : MyClassBase
{
	...
	// explicitamente indicar a ocultação do método da classe base
	public new void Func()
	{... }
}
```

- Métodos **não** são virtuais por defeito

```
public class MyClassBase
{
	...
	// indicar explicitamente que o método é virtual
	public virtual void Func()
	{ ... }
}

public class MyClassDeriv : MyClassBase
{
	...
	// indicar explicitamente a redefinição do método virtual da
	classe base
	public override void Func()
	{ ... }
}
```

- Aceder aos métodos da classe base

```
public class MyClassDeriv : MyClassBase
{
	...
	public override void Func()
	{
		base.Func();
		...
	}
}
```

2.3.7 Propriedades

- Sintaxe alternativa para acesso a membros de dados da classe mas com as vantagens dos métodos

```
public class Button : Control
{
	// atributo privado da classe
	private string caption;

	// propriedade pública
	public string Caption
	{
		// acessor ou getter
		get { return caption; }

		// mutator ou setter
		set { caption = value; Repaint(); }
	}
}
```

2.3.8 Objectos

- criação de objectos

```
// definição da classe
public class MyClass { ... }

// definição da variável
MyClass obj;

// criação do objecto
obj = new MyClass();
```

- aceder ao próprio objecto

```
// utilização da pseudovariavel
this.title = “ADAV”;
```

2.3.9 Arrays

- Suportados ao nível da biblioteca base de classes em System.Array

```
// declaração do vector
string[] vec;

// criação do vector
vec = new string[10];

// número de elementos pode ser dinâmico
vec = new string[n];
```

2.3.10 Ciclos

```
// repetição n vezes
for (int x = 0; i < vec.Length; i++)
      Console.WriteLine(vec[i]);


// repetição condicional
int i = 0;
while (i < vec.Length)
{
      Console.WriteLine(vec[i]);
      i++;
}


// enumeração
foreach (String x in vec)
      Console.WriteLine(x);
```

2.3.11 Condicionais

```
// teste de decisão
if (i < vec.Length)
      Console.WriteLine(vec[i]);
else
      Console.WriteLine(“Erro!!!”);
```

```
// teste múltiplo
switch (x)
{
      case 1: ...; break;
      case 2: ...; goto case 3; // fall trough explícito
      case 3: ...; break;
      default: ...; break;
}
```

2.3.12 Comentários em XML

- Suportados pelo Visual Studio .net
- Documentam o código (classes e métodos)

```
class MinhaClasse
{
      /// <summary>
      /// Returns the attribute with the given name and
      /// namespace</summary>
      /// <param name="name">The name of the attribute </param>
      /// <param name="ns">The namespace of the attribute, or null if
      /// the attribute has no namespace</param>
      /// <return> The attribute value, or null if the attribute
      /// does not exist</return>
      /// <seealso cref=" ="GetAttr(string)"/>
      public string GetAttr(string name, string ns) {
            ... ...
      }
}
```

2.4 Sites para consulta com informação adicional

MSDN Library

http://msdn.microsoft.com/library

Design Guidelines for Class Library Developers

http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

.net framework center

http://msdn.microsoft.com/netframework/

C#

http://msdn.microsoft.com/vcsharp/

ECMA

http://www.ecma-international.org/

Introduction to C# @ ECMA

http://www.ecma-international.org/activities/Languages/Introduction%20to%20Csharp.pdf

Common Language Infrastructure @ ECMA

http://www.ecma-international.org/activities/Languages/ECMA%20CLI%20Presentation.pdf

Using ADO.net

http://msdn.microsoft.com/netframework/using/understanding/data/default.aspx?pull=/library/en-us/dndotnet/html/usingadonet.asp

ASP.net

http://www.asp.net

Winforms

http://www.windowsforms.net/

Laboratório .net do ISEP/IPP

http://www.dei.isep.ipp.pt/labdotnet/

Open CLI

http://sourceforge.net/projects/ocl

Mono (.net @ Unix)

http://www.go-mono.com/

Anexo 3 Usar a plataforma Java em vez da plataforma .Net

3.1 Introdução

De acordo com as definições de componente apresentadas na secção 4.1, **um componente é uma unidade de distribuição em formato binário, que fornece serviços através de um contrato bem definido** (i.e., uma interface); por sua vez, **uma interface é um conjunto coerente de métodos** (métodos relacionados com um mesmo serviço e/ou trabalhando com entidades de um mesmo domínio). Um componente é utilizado por terceiros aos quais se dá o nome de "cliente", **um cliente é uma aplicação ou outro componente que faz uso dos serviços de um componente**.

Em java[15] para responder a estes requisitos temos (Tabela 5):

Tabela 5 - mapeamento entre requisitos para componentes e a tecnologia java

Requisito	Mapeamento .net
Unidade de distribuição binária	Ficheiro JAR
Fornecimento de serviços	Classes
Contrato explícito	Interfaces
Reutilização	Referência ao ficheiro JAR no *classpath*
Composição por terceiros	Composição e especialização (herança) de classes

Ter em atenção que na plataforma Java, um ficheiro JAR não é equivalente a um *assembly* na plataforma .net. O mecanismo de *loading* da máquina virtual java é baseado em classes enquanto que na CLR é baseado em *app domains* em que cada *app domain* carrega um ou mais *assemblies* que por sua vez definem classes.

Para desenvolver na plataforma Java devem fazer download do JDK e opcionalmente de um ambiente de desenvolvimento. Os exemplos apresentados neste capítulo foram desenvolvidos usando o JDK 1.5 (*standard edition*) e o NetBeans 4.1 que podem efectuar download em http://java.sun.com/j2se/1.5.0/download.jsp

[15] Para mais informação sobre como começar a usar a plataforma Java, consulte o tutorial de Java em http://java.sun.com/docs/books/tutorial/. Para informação sobre as convenções de codificação em Java consultar http://java.sun.com/blueprints/code/namingconventions.html

3.2 Guião de trabalho: Componente Aritmética com Factory

Seguindo as regras do Java, cada *package* corresponde a um directório no disco, pelo que devem criar uma estrutura semelhante à da Figura 50. Se usarem o NetBeans devem criar um projecto separado para cada componente da aplicação (Figura 51).

Figura 50 – estrutura de directórios e ficheiros para exemplo Aritmetica em Java

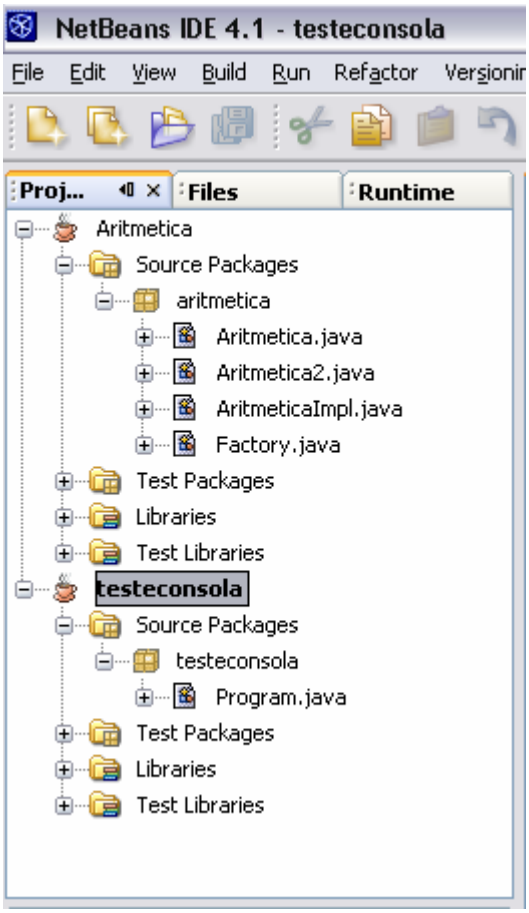

Figura 51 – estrutura de projectos no NetBeans para exemplo Aritmetica

3.2.1 Componente

aritmetica/Aritmetica.java

```
package aritmetica;

public interface Aritmetica
{
   float adicao(float op1, float op2);
   float subtracao(float op1, float op2);
   float multiplicacao(float op1, float op2);
   float divisao(float op1, float op2);
}
```

aritmetica/Aritmetica2.java

```
package aritmetica;

public interface Aritmetica2 extends Aritmetica
{
   double potencia(float base, int expoente);
   double raizQuadrada(float numero);
}
```

aritmetica/Factory.java

```
package aritmetica;

public class Factory
{
   public static Aritmetica Create() {
      return (Aritmetica)new AritmeticaImpl();
   }

   public static Aritmetica2 Create2() {
      return (Aritmetica2)new AritmeticaImpl();
   }
}
```

aritmetica/AritmeticaImpl.java

```
package aritmetica;

public class AritmeticaImpl implements Aritmetica, Aritmetica2
{
   public AritmeticaImpl() { }

   public float adicao(float op1, float op2)
   {
      return op1 + op2;
   }

   public float subtracao(float op1, float op2)
   {
      return op1 - op2;
   }
```

```
   public float multiplicacao(float op1, float op2)
   {
      return op1 * op2;
   }

   public float divisao(float op1, float op2) throws ArithmeticException
   {
      if (op2 != 0)
         return op1 / op2;
      else
         throw new ArithmeticException("tentou efectuar uma divisão por
zero");
   }

   public double potencia(float b, int expoente)
   {
      return Math.pow(b, expoente);
   }

   public double raizQuadrada(float n)
   {
      return Math.sqrt(n);
   }
}
```

3.2.2 Aplicação de teste de consola

testeconsola/Program.java

```
package testeconsola;

import aritmetica.Factory;
import aritmetica.Aritmetica;

public class Program
{
   public static void main(String[] args)
   {
      int op1 = 5;
      int op2 = 7;

      //declarar objecto para o serviço que pretendemos - interface
      Aritmetica comp;

      //criar objecto com implementação do serviço
      comp = Factory.Create();

      //invocar o serviço pretendido
      System.out.println("Adição(" + op1 + ", " + op2 + " => " +
                         comp.adicao(op1, op2));
   }
}
```

3.2.3 Compilação

NOTA: *caso tenham usado o NetBeans para criar o projecto e editar o código podem utilizar a opção de build e debug do NetBeans em vez de usar o JDK na linha de comando como em seguida se apresenta.*

1. para criar o componente

1.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ExAritmetica
```

1.2. compilar código fonte do componente de aritmética

```
javac –d c:\dev-java\bin aritmetica/*.java
```

1.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

1.4. criar JAR

```
jar cvf c:\dev-java\lib\aritmetica.jar aritmetica/*.class
```

2. para criar o programa de teste

2.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ExAritmetica
```

2.2. compilar programa de teste

```
javac -cp c:\dev-java\lib\aritmetica.jar -d c:\dev-java\bin
     testeconsola/*.java
```

2.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

2.4. executar

```
java testeconsola.Program
```

3.3 Guião de trabalho: Carregamento dinâmico

Seguindo as regras do Java, cada *package* corresponde a um directório no disco, pelo que devem criar uma estrutura semelhante à da Figura 52. Se usarem o NetBeans devem criar um projecto separado para cada componente da aplicação (Figura 53).

Figura 52 – estrutura de ficheiros e directórios para exemplo de carregamento dinâmico em Java

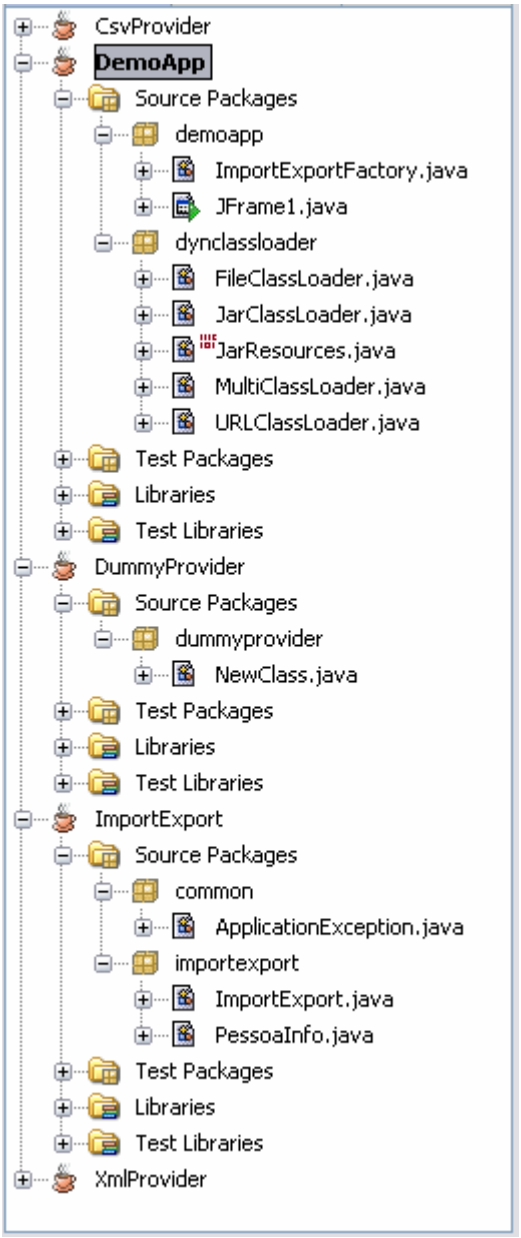

Figura 53 – estrutura de projectos no NetBeans para exemplo de carregamento dinâmico

3.3.1 Componente com interface comum

common/ApplicationException.java

```
package common;

public class ApplicationException extends Exception{

   /** Creates a new instance of ApplicationException */
   public ApplicationException() {
      super();
   }

   public ApplicationException(String msg) {
      super(msg);
   }
```

```
  public ApplicationException(String msg, Exception inner) {
    super(msg, inner);
  }
}
```

importexport/ImportExport.java

```
package importexport;

import common.ApplicationException;
import java.util.List;

public interface ImportExport
{
  void doExport(List<PessoaInfo> data)
              throws java.io.IOException, ApplicationException;
  List<PessoaInfo> doImport()
              throws java.io.IOException, ApplicationException;
}
```

importexport/PessoaInfo.java

```
package importexport;

public class PessoaInfo
{
  public PessoaInfo() { }

  public PessoaInfo(String n, int i)
  {
    Nome = n;
    Idade = i;
  }

  public String Nome;
  public int Idade;

  public String toString()
  {
    return Nome + "(" + Idade + ")";
  }
}
```

3.3.2 Provider de teste (dummy)

dummyprovider/NewClass.java

```
package dummyprovider;

import importexport.ImportExport;
import importexport.PessoaInfo;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.List;
```

```
public class NewClass implements ImportExport
{
   /** Creates a new instance of NewClass */
   public NewClass() { }

   public void doExport(List<PessoaInfo> data)
               throws java.io.IOException
   {
      Writer wr = new BufferedWriter(
                  new FileWriter("c:\\temp\\dummy-export.txt")
                  );
      for (PessoaInfo pi : data)
      {
         wr.write(pi.toString());
         wr.newLine();
      }
      wr.close();
   }

   public List<PessoaInfo> doImport()
             throws java.io.IOException
   {
      List<PessoaInfo> ret = new ArrayList<PessoaInfo>();

      ret.add(new PessoaInfo("joao", 29));
      ret.add(new PessoaInfo("carla", 25));
      ret.add(new PessoaInfo("margarida", 32));

      return ret;
   }
}
```

3.3.3 Aplicação cliente

demoapp/JFrame1.java

NOTA: *Criada usando NetBeans 4.1 e JDK 1.5.*

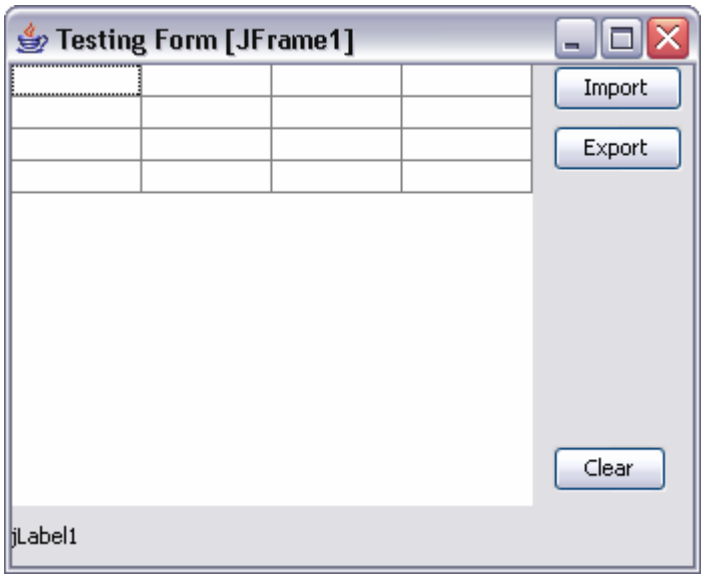

Figura 54 - GUI da aplicação demo de carregamento dinâmico em Java

```
package demoapp;

import common.ApplicationException;
import importexport.ImportExport;
import importexport.PessoaInfo;
import java.util.List;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
/*
 * JFrame1.java
 *
 * Created on 20 de Outubro de 2005, 18:02
 */

public class JFrame1 extends javax.swing.JFrame {

  /** Creates new form JFrame1 */
  public JFrame1() {
    initComponents();
  }

  /** This method is called from within the constructor to
   * initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is
   * always regenerated by the Form Editor.
   */
  // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
  private void initComponents() {
    jtDados = new javax.swing.JTable();
    btnImport = new javax.swing.JButton();
    btnExport = new javax.swing.JButton();
    btnClear = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();

    getContentPane().setLayout(
                new org.netbeans.lib.awtextra.AbsoluteLayout());


  setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    jtDados.setModel(new javax.swing.table.DefaultTableModel(
      new Object [][] {
        {null, null, null, null},
        {null, null, null, null},
        {null, null, null, null},
        {null, null, null, null}
      },
      new String [] {
        "Title 1", "Title 2", "Title 3", "Title 4"
      }
    ));
    getContentPane().add(jtDados,
    new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 260, 220)
    );

    btnImport.setText("Import");
    btnImport.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnImportActionPerformed(evt);
      }
    });
```

```
    getContentPane().add(btnImport,
    new org.netbeans.lib.awtextra.AbsoluteConstraints(270, 0, -1, -1)
    );

    btnExport.setLabel("Export");
    btnExport.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnExportActionPerformed(evt);
      }
    });

    getContentPane().add(btnExport,
    new org.netbeans.lib.awtextra.AbsoluteConstraints(270, 30, -1, -1)
    );

    btnClear.setLabel("Clear");
    btnClear.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnClearActionPerformed(evt);
      }
    });

    getContentPane().add(btnClear,
  new org.netbeans.lib.awtextra.AbsoluteConstraints(270, 190, -1, -1)
    );

    jLabel1.setText("jLabel1");
    getContentPane().add(jLabel1,
    new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 220, 340, 30)
    );

    pack();
  }
  // </editor-fold>

  private void btnExportActionPerformed(java.awt.event.ActionEvent evt)
  {
    try
    {
      ImportExportFactory fact = new ImportExportFactory();
      ImportExport prov = fact.CreateProvider();
      prov.doExport(
((MeuDataSource<PessoaInfo>)jtDados.getModel()).getData() );
    }
    catch (ApplicationException ex)
    {
      jLabel1.setText(ex.getMessage());
    }
    catch (java.io.IOException ex)
    {
      jLabel1.setText(ex.getMessage());
    }
  }

  private void btnClearActionPerformed(java.awt.event.ActionEvent evt)
  {
    jtDados.setModel(new DefaultTableModel(0, 2));
  }

  private void btnImportActionPerformed(java.awt.event.ActionEvent evt)
  {
```

```
    try
    {
      ImportExportFactory fact = new ImportExportFactory();
      ImportExport prov = fact.CreateProvider();
      List<PessoaInfo> dados = prov.doImport();

      jtDados.setModel(new MeuDataSource<PessoaInfo>(dados) {
           public int getColumnCount() { return 2; }
           public boolean isCellEditable(int row, int col)
           { return true; }
           public void setAttribute(Object attr, int row, int col){
             PessoaInfo item = this.getData().get(row);
             switch (col)
             {
               case 0:
                 item.Nome = (String)attr;
                 break;
               case 1:
                 try
                 {
                   item.Idade = Integer.parseInt((String)attr);
                 }
                 catch (NumberFormatException ex)
                 {
                 }
                 break;
             }
           }
           public Object getAttribute(PessoaInfo item, int col) {
             switch (col)
             {
               case 0:
                 return item.Nome;
               case 1:
                 return item.Idade;
             }
             return null;
           }
         }
      );
    }
    catch (ApplicationException ex)
    {
      jLabel1.setText(ex.getMessage());
    }
    catch (java.io.IOException ex)
    {
      jLabel1.setText(ex.getMessage());
    }
  }

  private abstract class MeuDataSource<T> extends AbstractTableModel
  {
    private List<T> source;

    public MeuDataSource(List<T> source)
    {
      this.source = source;
    }

    public List<T> getData() {
```

```
        return source;
      }

      public int getRowCount() { return source.size();}
      public Object getValueAt(int row, int col)
      {
        for (int i=0; i < source.size(); i++)
        {
          if (i == row)
          {
            return getAttribute(source.get(i), col);
          }
        }
        return null;
      }

    public void setValueAt(Object value, int row, int col) {
      setAttribute(value, row, col);
      fireTableCellUpdated(row, col);
    }

      public abstract void setAttribute(Object item, int row, int col);
       public abstract Object getAttribute(T item, int col);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
          new JFrame1().setVisible(true);
        }
      });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnClear;
    private javax.swing.JButton btnExport;
    private javax.swing.JButton btnImport;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTable jtDados;
    // End of variables declaration
}
```

demoapp/ImportExportFactory.java

```
package demoapp;

import common.ApplicationException;
import dynclassloader.FileClassLoader;
import dynclassloader.JarClassLoader;
import dynclassloader.URLClassLoader;
import importexport.ImportExport;
import java.io.FileInputStream;
import java.util.Properties;

public class ImportExportFactory {

    public ImportExportFactory() {
```

```
  }

  public ImportExport CreateProvider() throws ApplicationException
  {
    try
    {
      // create and load default properties
      Properties defaultProps = new Properties();
      FileInputStream in =
             new FileInputStream("c:\\temp\\demoapp.properties");
      defaultProps.load(in);
      in.close();

      // ler valores de configuração
      String classname =
               defaultProps.getProperty("ImpExpProviderClass");
      if (classname=="")
        throw new ApplicationException("falta configurar provider");
      String jarname =
               defaultProps.getProperty("ImpExpProviderSource");
      if (jarname=="")
        throw new ApplicationException("falta configurar provider");

      // carregar a classe dinamicamente
      //Class cl = Class.forName(classname);

      // Create the class loader
      ClassLoader loader = null;
      if (jarname.toLowerCase().endsWith(".jar"))
      {
        loader = new JarClassLoader (jarname);
      }
      else if (jarname.toLowerCase().endsWith("\\"))
      {
        loader = new FileClassLoader(jarname);
      }
      else
      {
        loader = new URLClassLoader(jarname);
      }


      /* Load the class from the jar file and resolve it. */
      Class cl = loader.loadClass(classname);

      //cria uma nova instancia usando construtor por defeito
      Object obj = cl.newInstance();

      if (obj instanceof ImportExport)
        return (ImportExport)obj;
      else
        throw new ApplicationException("classe não implementa
interface adequado");
      }
    catch (java.io.FileNotFoundException ex)
    {
      throw new ApplicationException("ficheiro não encontrado", ex);
    }
    catch (java.io.IOException ex)
    {
      throw new ApplicationException("erro de IO", ex);
```

```
      }
      catch (IllegalAccessException ex)
      {
        throw new ApplicationException("não foi possivel criar objecto
provider", ex);
      }
      catch (InstantiationException ex)
      {
        throw new ApplicationException("não foi possivel criar objecto
provider", ex);
      }
      catch (ClassNotFoundException ex)
      {
        throw new ApplicationException("classe  provider não
encontrado", ex);
      }
    }
}
```

dynclassloader/MultiClassLoader.java

```
// retirado de
// http://www.javaworld.com/javaworld/javatips/jw-javatip39.html
package dynclassloader;

import java.util.Hashtable;


/**
 * A simple test class loader capable of loading from
 * multiple sources, such as local files or a URL.
 *
 * This class is derived from an article by Chuck McManis
 * http://www.javaworld.com/javaworld/jw-10-1996/indepth.src.html
 * with large modifications.
 *
 * Note that this has been updated to use the non-deprecated version of
 * defineClass() -- JDM.
 *
 * @author Jack Harich - 8/18/97
 * @author John D. Mitchell - 99.03.04
 */
public abstract class MultiClassLoader extends ClassLoader {

//---------- Fields --------------------------------------
private Hashtable classes = new Hashtable();
private char    classNameReplacementChar;

protected boolean   monitorOn = false;
protected boolean   sourceMonitorOn = true;

//---------- Initialization ------------------------------
public MultiClassLoader() {
}
//---------- Superclass Overrides ------------------------
/**
 * This is a simple version for external clients since they
 * will always want the class resolved before it is returned
 * to them.
 */
```

```
public Class loadClass(String className) throws ClassNotFoundException
{
  return (loadClass(className, true));
}
//---------- Abstract Implementation ---------------------
public synchronized Class loadClass(String className,
     boolean resolveIt) throws ClassNotFoundException {

  Class   result;
  byte[]  classBytes;
  monitor(">> MultiClassLoader.loadClass(" + className + ", " +
resolveIt + ")");

  //----- Check our local cache of classes
  result = (Class)classes.get(className);
  if (result != null) {
    monitor(">> returning cached result.");
    return result;
  }

  //----- Check with the primordial class loader
  try {
    result = super.findSystemClass(className);
    monitor(">> returning system class (in CLASSPATH).");
    return result;
  } catch (ClassNotFoundException e) {
    monitor(">> Not a system class.");
  }

  //----- Try to load it from preferred source
  // Note loadClassBytes() is an abstract method
  classBytes = loadClassBytes(className);
  if (classBytes == null) {
    throw new ClassNotFoundException();
  }

  //----- Define it (parse the class file)
  result = defineClass(className, classBytes, 0, classBytes.length);
  if (result == null) {
    throw new ClassFormatError();
  }

  //----- Resolve if necessary
  if (resolveIt) resolveClass(result);

  // Done
  classes.put(className, result);
  monitor(">> Returning newly loaded class.");
  return result;
}
//---------- Public Methods ------------------------------
/**
 * This optional call allows a class name such as
 * "COM.test.Hello" to be changed to "COM_test_Hello",
 * which is useful for storing classes from different
 * packages in the same retrival directory.
 * In the above example the char would be '_'.
 */
public void setClassNameReplacementChar(char replacement) {
  classNameReplacementChar = replacement;
}
```

```
//---------- Protected Methods ---------------------------
protected abstract byte[] loadClassBytes(String className);

protected String formatClassName(String className) {
  if (classNameReplacementChar == '\u0000') {
    // '/' is used to map the package to the path
    return className.replace('.', '/') + ".class";
  } else {
    // Replace '.' with custom char, such as '_'
    return className.replace('.',
      classNameReplacementChar) + ".class";
  }
}
protected void monitor(String text) {
  if (monitorOn) print(text);
}
//--- Std
protected static void print(String text) {
  System.out.println(text);
}

} // End class
```

dynclassloader/FileClassLoader.java

```
// retirado de
// http://www.javaworld.com/javaworld/javatips/jw-javatip39.html
package dynclassloader;

import java.io.ByteArrayInputStream;
import java.io.FileInputStream;
import java.io.InputStream;

/**
 * Loads class bytes from a file.
 *
 * @author Jack Harich - 8/30/97
 */
public class FileClassLoader extends MultiClassLoader {

//---------- Private Fields -----------------------------
private StringfilePrefix;

//---------- Initialization -----------------------------
/**
 * Attempts to
 * load from a local file using the relative "filePrefix",
 * ie starting at the current directory. For example
 * @param filePrefix could be "webSiteClasses\\site1\\".
 */
public FileClassLoader(String filePrefix) {
  this.filePrefix = filePrefix;
}
//---------- Abstract Implementation --------------------
protected byte[] loadClassBytes(String className) {

  className = formatClassName(className);
  if (sourceMonitorOn) {
    print(">> from file: " + className);
  }
```

```
   byte result[];
   String fileName = filePrefix + className;
   try {
      FileInputStream inStream = new FileInputStream(fileName);
      // *** Is available() reliable for large files?
      result = new byte[inStream.available()];
      inStream.read(result);
      inStream.close();
      return result;

   } catch (Exception e) {
      // If we caught an exception, either the class
      // wasn't found or it was unreadable by our process.
      print("### File '" + fileName + "' not found.");
      return null;
   }
}

} // End class
```

dynclassloader/URLClassLoader.java

```
// retirado de
// http://www.javaworld.com/javaworld/javatips/jw-javatip39.html
package dynclassloader;

import java.net.URL;
import java.net.URLConnection;
import java.io.InputStream;

/**
 * Loads class bytes from a URL, such as
 * "http://www.mindspring.com/~happyjac/".
 *
 * @author Jack Harich - 8/30/97
 */
public class URLClassLoader extends MultiClassLoader {

//---------- Private Fields -----------------------------
private StringurlString;

//---------- Initialization -----------------------------
public URLClassLoader(String urlString) {
   this.urlString = urlString;
}
//---------- Abstract Implementation --------------------
protected byte[] loadClassBytes(String className) {

   className = formatClassName(className);
   try {
      URL url = new URL(urlString + className);
      URLConnection connection = url.openConnection();
      if (sourceMonitorOn) {
         print("Loading from URL: " + connection.getURL() );
      }
      monitor("Content type is: " + connection.getContentType());

      InputStream inputStream = connection.getInputStream();
      int length = connection.getContentLength();
      monitor("InputStream length = " + length); // Failure if -1
```

```
      byte[] data = new byte[length];
      inputStream.read(data); // Actual byte transfer
      inputStream.close();
      return data;

    } catch(Exception ex) {
      print("### URLClassLoader.loadClassBytes() - Exception:");
      ex.printStackTrace();
      return null;
    }
}

} // End class
```

dynclassloader/JarResources.java

```
// retirado de
// http://www.javaworld.com/javaworld/javatips/jw-javatip49.html
package dynclassloader;

import java.io.*;
import java.util.*;
import java.util.zip.*;

/**
 * JarResources: JarResources maps all resources included in a
 * Zip or Jar file. Additionaly, it provides a method to extract one
 * as a blob.
 */
public final class JarResources
   {

   // external debug flag
   public boolean debugOn=false;

   // jar resource mapping tables
   private Hashtable htSizes=new Hashtable();
   private Hashtable htJarContents=new Hashtable();

   // a jar file
   private String jarFileName;

   /**
     * creates a JarResources. It extracts all resources from a Jar
     * into an internal hashtable, keyed by resource names.
     * @param jarFileName a jar or zip file
     */
   public JarResources(String jarFileName)
   {
   this.jarFileName=jarFileName;
   init();
   }

   /**
     * Extracts a jar resource as a blob.
     * @param name a resource name.
     */
   public byte[] getResource(String name)
   {
```

```
return (byte[])htJarContents.get(name);
}

/** initializes internal hash tables with Jar file resources.  */
private void init()
{
try
   {
   // extracts just sizes only.
   ZipFile zf=new ZipFile(jarFileName);
   Enumeration e=zf.entries();
   while (e.hasMoreElements())
   {
   ZipEntry ze=(ZipEntry)e.nextElement();

   if (debugOn)
      {
      System.out.println(dumpZipEntry(ze));
      }

   htSizes.put(ze.getName(),new Integer((int)ze.getSize()));
   }
   zf.close();

   // extract resources and put them into the hashtable.
   FileInputStream fis=new FileInputStream(jarFileName);
   BufferedInputStream bis=new BufferedInputStream(fis);
   ZipInputStream zis=new ZipInputStream(bis);
   ZipEntry ze=null;
   while ((ze=zis.getNextEntry())!=null)
   {
   if (ze.isDirectory())
      {
      continue;
      }

   if (debugOn)
      {
      System.out.println("ze.getName()="+ze.getName()+
              ","+"getSize()="+ze.getSize() );
      }

   int size=(int)ze.getSize();
   // -1 means unknown size.
   if (size==-1)
      {
      size=((Integer)htSizes.get(ze.getName())).intValue();
      }

   byte[] b=new byte[(int)size];
   int rb=0;
   int chunk=0;
   while (((int)size - rb) > 0)
      {
      chunk=zis.read(b,rb,(int)size - rb);
      if (chunk==-1)
      {
      break;
      }
      rb+=chunk;
      }
```

```
   // add to internal resource hashtable
   htJarContents.put(ze.getName(),b);

   if (debugOn)
      {
      System.out.println( ze.getName()+"  rb="+rb+
           ",size="+size+
           ",csize="+ze.getCompressedSize() );
      }
   }
   }
catch (NullPointerException e)
   {
   System.out.println("done.");
   }
catch (FileNotFoundException e)
   {
   e.printStackTrace();
   }
catch (IOException e)
   {
   e.printStackTrace();
   }
}

/**
  * Dumps a zip entry into a string.
  * @param ze a ZipEntry
  */
private String dumpZipEntry(ZipEntry ze)
{
StringBuffer sb=new StringBuffer();
if (ze.isDirectory())
   {
   sb.append("d ");
   }
else
   {
   sb.append("f ");
   }

if (ze.getMethod()==ZipEntry.STORED)
   {
   sb.append("stored   ");
   }
else
   {
   sb.append("defalted ");
   }

sb.append(ze.getName());
sb.append("\t");
sb.append(""+ze.getSize());
if (ze.getMethod()==ZipEntry.DEFLATED)
   {
   sb.append("/"+ze.getCompressedSize());
   }

return (sb.toString());
}
```

```
   } // End of JarResources class.
```

dynclassloader/JarClassLoader.java

```
// retirado de
// http://www.javaworld.com/javaworld/javatips/jw-javatip70.html
package dynclassloader;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   Copyright (c) Non, Inc. 1999 -- All Rights Reserved

PACKAGE:  JavaWorld
FILE:     JarClassLoader.java

AUTHOR:   John D. Mitchell, Mar  3, 1999

REVISION HISTORY:
   Name Date   Description
   ---- ----   -----------
   JDM  99.03.03    Initial version.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/


/**
 ** JarClassLoader provides a minimalistic ClassLoader which shows
 ** how to instantiate a class which resides in a .jar file.
 **
 ** @author John D. Mitchell, Non, Inc., Mar  3, 1999
 **
 ** @version 0.5
 **
 **/

public class JarClassLoader extends MultiClassLoader
   {
   private JarResources jarResources;

   public JarClassLoader (String jarName)
   {
   // Create the JarResource and suck in the .jar file.
   jarResources = new JarResources (jarName);
   }

   protected byte[] loadClassBytes (String className)
   {
   // Support the MultiClassLoader's class name munging facility.
   className = formatClassName (className);

   // Attempt to get the class data from the JarResource.
   return (jarResources.getResource (className));
   }
} // End of Class JarClassLoader.
```

demoapp.properties

```
ImpExpProviderClass=xmlprovider.NewClass
ImpExpProviderSource=C:\\DEV-Java\\lib\\XmlProvider.jar
```

3.3.4 Provider CSV

csvprovider/NewClass.java

```
package csvprovider;

import common.ApplicationException;
import importexport.ImportExport;
import importexport.PessoaInfo;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Paulo Sousa
 */
public class NewClass implements ImportExport
{

   /** Creates a new instance of NewClass */
   public NewClass() {
   }

   public void doExport(List<PessoaInfo> data)
                     throws java.io.IOException
   {
      BufferedWriter wr = new BufferedWriter(
                    new FileWriter("c:\\temp\\csv-data.csv")
                    );
      for (PessoaInfo pi : data)
      {
         wr.write(pi.Nome);
         wr.write(", ");
         wr.write(String.valueOf(pi.Idade));
         wr.newLine();
      }
      wr.close();
   }

   public List<PessoaInfo> doImport()
                 throws java.io.IOException, ApplicationException
   {
      List<PessoaInfo> ret = new ArrayList<PessoaInfo>();

      try
      {
         BufferedReader rd = new BufferedReader(
                           new FileReader("c:\\temp\\csv-data.csv")
                           );
         String line=null;
         while ((line = rd.readLine()) != null)
```

```
        {
          String[] parts = line.split(",");
          ret.add(new PessoaInfo(parts[0],
                                 Integer.parseInt(parts[1].trim()))
                  );
        }
      }
      catch (ArrayIndexOutOfBoundsException ex)
      {
        throw new ApplicationException("ficheiro no formato inválido",
ex);
      }
      catch (NumberFormatException ex)
      {
        throw new ApplicationException("ficheiro no formato inválido",
ex);
      }
      return ret;
    }
}
```

3.3.5 Provider XML

xmlprovider/NewClass.java

```
package xmlprovider;

import common.ApplicationException;
import importexport.ImportExport;
import importexport.PessoaInfo;
import java.util.List;

// XML support
import java.io.File;
import java.util.ArrayList;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;


/**
 *
 * @author Paulo Sousa
 */
public class NewClass implements ImportExport
{

  /** Creates a new instance of NewClass */
  public NewClass() {
  }
```

```
// This method writes a DOM document to a file
private void writeXmlFile(Document doc, String filename) {
   try {
     // Prepare the DOM document for writing
     Source source = new DOMSource(doc);

     // Prepare the output file
     File file = new File(filename);
     Result result = new StreamResult(file);

     // Write the DOM document to the file
     Transformer xformer =
            TransformerFactory.newInstance().newTransformer();
     xformer.transform(source, result);
   } catch (TransformerConfigurationException e) {
   } catch (TransformerException e) {
   }
}

public void doExport(List<PessoaInfo> data)
                 throws java.io.IOException, ApplicationException
{
   try
   {
     DocumentBuilderFactory docBuilderFactory =
                        DocumentBuilderFactory.newInstance();
     DocumentBuilder docBuilder =
                        docBuilderFactory.newDocumentBuilder();
     Document doc = docBuilder.newDocument();

     Element root = doc.createElement("pessoas");
     doc.appendChild(root);

     for(PessoaInfo pi : data)
     {
       Element newChild = doc.createElement("pessoa");
       newChild.setAttribute("nome", pi.Nome);
       newChild.setAttribute("idade", String.valueOf(pi.Idade));

       root.appendChild(newChild);
     }

     writeXmlFile(doc, "c:\\temp\\xml-data.xml");
   }
   catch (javax.xml.parsers.ParserConfigurationException ex)
   {
     throw new ApplicationException("erro parser XML", ex);
   }
}

public List<PessoaInfo> doImport()
                     throws java.io.IOException, ApplicationException
{
   List<PessoaInfo> ret = new ArrayList();

   try {
   DocumentBuilderFactory docBuilderFactory =
                           DocumentBuilderFactory.newInstance();
   DocumentBuilder docBuilder =
                           docBuilderFactory.newDocumentBuilder();
```

```
      Document doc = docBuilder.parse (
                        new File("c:\\temp\\xml-data.xml")
                        );

      // normalize text representation
      doc.getDocumentElement ().normalize ();

      NodeList listOfPersons = doc.getElementsByTagName("pessoa");
      int totalPersons = listOfPersons.getLength();
      for(int s=0; s < listOfPersons.getLength() ; s++)
      {
        Node personNode = listOfPersons.item(s);
        NamedNodeMap attrs = personNode.getAttributes();
        Node nomeNode = attrs.getNamedItem("nome");
        String nome = nomeNode.getNodeValue();

        Node idadeNode = attrs.getNamedItem("idade");
        String idade = idadeNode.getNodeValue();

        ret.add(new PessoaInfo(nome,  Integer.parseInt(idade)));
      }
      }catch (SAXParseException err) {
        throw new ApplicationException("erro Parser SAX", err);
      }catch (SAXException e) {
        throw new ApplicationException("erro SAX", e);
      }
      catch (javax.xml.parsers.ParserConfigurationException ex)
      {
        throw new ApplicationException("erro parser XML", ex);
      }
    return ret;
    }
}
```

3.3.6 Compilação

NOTA: *caso tenham usado o NetBeans para criar o projecto e editar o código podem utilizar a opção de build e debug do NetBeans em vez de usar o JDK na linha de comando como em seguida se apresenta.*

1. para criar o componente ImportExport

1.2. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ExCarrDinamico
```

1.3. compilar código fonte do componente

```
javac –d c:\dev-java\bin importexport/*.java common/*.java
```

1.4. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

1.5. criar JAR

```
jar cvf ..\lib\importexport.jar importexport/*.class
common/*.class
```

2. para criar o componente DumyProvider

2.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ExCarrDinamico
```

2.2. compilar código fonte do componente

```
javac –d c:\dev-java\bin dummyprovider/*.java
```

2.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

2.4. criar JAR

```
jar cvf ..\lib\dummyprovider.jar dummyprovider/*.class
```

3. para criar o componente CsvProvider

3.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ADAV\ExCarregamentoDyn
```

3.2. compilar código fonte do componente

```
javac –d c:\dev-java\bin csvprovider/*.java
```

3.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

3.4. criar JAR

```
jar cvf ..\lib\csvprovider.jar  csvprovider/*.class
```

4. para criar o componente XmlProvider

4.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ADAV\ExCarregamentoDyn
```

4.2. compilar código fonte do componente

```
javac –d c:\dev-java\bin xmlprovider/*.java
```

4.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

4.4. criar JAR

```
jar cvf ..\lib\xmlprovider.jar  xmlprovider/*.class
```

5. para criar o programa de teste

5.1. mudar o directório de trabalho para o directório de topo do código fonte

```
CD c:\dev-java\ADAV\ExCarregamentoDyn
```

5.2. compilar programa de teste

```
javac -cp c:\dev-java\lib\importexport.jar;C:\Progra~1
\netbeans-4.1\ide5\modules\ext\AbsoluteLayout.jar -d c:\dev-
java\bin   demoapp/*.java  dynclassloader/*.class
```

5.3. mudar directório de trabalho para o directório de output da compilação

```
CD c:\dev-java\bin
```

5.4. executar

```
java –cp .;importexport.jar;c:\progra~1\netbeans-
4.1\ide5\modules\ext\AbsoluteLayout.jar demoapp.JFrame1
```