Escola Profissional da Horta

Curso Técnico de Informática / Gestão - 3º Ano

2007-2008

LPRO

Linguagens de Programação

FICHEIROS E STREAMS EM C#

Fevereiro de 08

Bruno Moniz

Introdução

- Na plataforma .NET os ficheiros são representados por objectos.
- No entanto, ao contrário de linguagens como o C em que as operações de leitura/escrita em ficheiros são directas, em .NET a leitura/escrita de ficheiros é abstraída em termos de *streams*.
- Uma ***stream*** representa um fluxo de informação. Este fluxo de informação "vem de algum lado e vai para algum lado".
- Todas as classes relacionadas com a entrada/saída de dados encontram-se no espaço de nomes ***System.IO***.
  - *using System.IO;*

Gestão do sistema de ficheiros

- É necessário usar diversas classes para manipularmos o sistema de ficheiros. Estas classes representam estruturas como ficheiros e directórios e a informação a eles associada.
  - **FileSystemInfo**
    Classe base que representa qualquer objecto (ficheiro ou directório) no sistema de ficheiros;
  - **FileInfo** e **File**
    Permitem examinar e manipular ficheiros em disco. O FileInfo permite criar um objecto para podermos realizar várias opções sobre o mesmo ficheiro enquanto o File apenas possui métodos estáticos;
  - **DirectoryInfo** e **Directory**
    Igual ao caso anterior mas relacionado com directórios;
  - **Path**
    Classe utilitária que permite realizar operações sobre cadeias de caracteres que representa nomes de ficheiros.

Exemplo do File e FileInfo

- **File**

```
File.Move(@"c:\temp\xpto.txt", @"c:\temp\xpti.txt");
```

- **FileInfo**

```
FileInfo xpto = new FileInfo(@"c:\temp\xpto.txt");
xpto.MoveTo(@"c:\temp\xpti.txt");
```

Exemplo do File e FileInfo

- Uma questão importante, quando estamos a manipular um ficheiro, é verificar se ele existe. Podemos conseguir isso através da propriedade **FileInfo.Exists** ou **File.Exists()**;

```
FileInfo xpto = new FileInfo(@"c:\temp\xpto.txt");
If(xpto.Exists)
        xpto.MoveTo(@"c:\temp\xpti.txt");
```

Excepções nas operações com ficheiros e directórios

- Algumas das operações com ficheiros/directórios podem lançar excepções que devem ser tratadas. Exemplos:
  - Ficheiro pode ter sido apagado depois de termos verificado se ele existe;
  - Podemos não ter permissões para aceder ao ficheiro/directório;
  - Acontecer um erro ao tentar aceder ao ficheiro;
  - …
- A maior parte das excepções lançadas nas operações com ficheiros/directórios derivam de **System.IO.IOException**.
- Para tratar destas excepções deve proteger-se o código com um bloco **try…catch**

Excepções nas operações com ficheiros e directórios - Exemplo

```
try
{
        FileInfo xpto = new FileInfo(@"c:\temp\xpto.txt");
        if (xpto.Exists)
                xpto.MoveTo(@"c:\temp\xpti.txt");
        else
                Console.WriteLine(@"O ficheiro c:\temp\xpto.txt não existe!");
}
catch (Exception e)
{
        Console.WriteLine("Problema ao mover o ficheiro: {0}", e.Message);
}
```

- Note-se que o facto de estarmos a apanhar as excepções não invalida que se faça um teste explícito à existência do ficheiro.
- **As excepções devem ser utilizadas em situações de erro e não em situações em que um simples teste basta**

Propriedades e métodos disponíveis nas classes FileInfo e DirectoryInfo

- **Ver texto de apoio sobre Ficheiros e Streams:**
  - **FileInfo**
    páginas 285 e 286
  - **DirectoryInfo**
  - **Páginas 286 e 287**

Classe Path

- A classe **Path** dispõe apenas de métodos estáticos permitindo ao programador manipular, de uma forma simples, cadeias de caracteres que representem ficheiros ou directórios.
- Com a classe Path podemos fazer operações do género:
  - Verificar o nome de um ficheiro;
  - Verificar a extensão do ficheiro;
  - Alterar a extensão de um ficheiro;
  - …

```
string novoNome = Path.ChangeExtension(@"xpto.txt","doc");
```

```
Console.WriteLine("Caracteres inválidos: {0}", Path.InvalidPathChars);
```

- **Propriedades e métodos disponíveis na classe Path:**
  - **Ver texto de apoio sobre Ficheiros e Streams, página 288**

Revisões

- Todas as classes relacionadas com dispositivos de entrada e saída encontram-se em System.IO;
- A leitura/escrita para ficheiros é abstraída em termos de streams;
- Uma stream representa um canal de comunicação de/para uma certa fonte de informação;
- Para manipular ficheiros utilizam-se as classes File ou FileInfo. Ambas possuem a mesma funcionalidade. No entanto, File apenas dispõe de métodos estáticos, enquanto FileInfo representa um ficheiro em particular;

Revisões

- Para manipular directórios utilizam-se as classe Directory ou DirectoryInfo. A situação é idêntica ao ponto anterior;
- Antes de fazer uma manipulação de um objecto de um sistema de ficheiros, é conveniente verificar se este existe utilizando a propriedade FileInfo.Exists ou DirectortInfo.Exists;
- Ao realizar uma operação que envolva um objecto no sistema de ficheiros podem ocorrer diversos problemas. Pode ocorrer o levantamento de excepções que devem ser tratadas;
- A classe Path permite manipular de forma simples cadeias de caracteres que representam ficheiros e directórios.

Leitura/Escrita de ficheiros

- O conceito de ***stream*** é muito simples e, ao mesmo tempo, muito poderoso.
  - Uma **stream de entrada** representa a fonte de informação;
  - Uma **stream de saída** representa algo que envia informação para determinado local.
- Na plataforma .NET já existem classes para manipulação da informação. Basicamente, para tratar de **ficheiros de texto**, utilizam-se as classes **StreamReader** e **StreamWriter** e, para tratar de **ficheiros binários**, as classes **BinaryReader** e **BinaryWriter**.

Hierarquia de *streams*

- Na hierarquia de *streams* da plataforma .NET, podemos encontrar dois tipos distintos de *streams*:
  - ***Streams* que representam fontes de informação ou consumidores de informação.** Este tipo de *streams* constitui a base à qual as outras *streams* se ligam; (ex: FileStream e MemoryStream)
  - ***Streams que utilizam streams base para fazer tratamento dos dados que estas disponibilizam*.** Leva como parâmetro no seu construtor uma *stream* do tipo básico. (ex: StreamReader e BinaryReader).

Exemplo

```
try {
        FileStream ficheiro = new FileStream(@"xpto.txt", FileMode.Create);  1
        StreamWriter writer = new StreamWriter(ficheiro);
        write.Write("Hello");
        writer.Close();
}
catch (Exception e) {
        Console.WriteLine(e.Message);
}
```

- A expressão em 1 cria uma *stream* básica associada ao ficheiro "xpto.txt". Esta *stream* apenas permite operações como ler e escrever blocos de bytes.
- Para conseguirmos escrever texto no ficheiro, temos que encapsular esta *stream* numa outra mais poderosa: *StreamWriter*. No construtor desta *stream* passamos por parâmetro a *stream* associada ao ficheiro.

A classe *FileStream*

- A classe *FileStream* constitui a abstracção mínima para aceder a um ficheiro;
- O seu construtor permite associar um objecto deste tipo a um ficheiro, abrindo-o para leitura, escrita ou leitura e escrita, assim como com um certo modo de abertura, ou seja, escrita/leitura no fim do ficheiro, no inicio do ficheiro, etc);
- Existem vários construtores para o *FileStream* mas vamos só analisar o principal:

FileStream(string *fich*, FileMode *modAbertura*, FileAccess *modAcesso*);

FileMode e FileAccess

Valor	Descrição
FileMode.Append	Caso o ficheiro não exista, cria o ficheiro. Caso exista, abre-o sendo as escritas realizadas no fim deste.
FileMode.Create	Caso o ficheiro não exista, cria o ficheiro. Caso exista, o ficheiro é aberto mas o conteúdo apagado.
FileMode.CreateNew	Caso o ficheiro não exista, cria o ficheiro. Caso exista, é lançada uma excepção.
FileMode.Open	Abre o ficheiro. Caso não exista, lança uma Excepção.
FileMode.OpenOrCreate	Abre o ficheiro. Caso não exista, o ficheiro é criado.
FileMode.Truncate	Abre um ficheiro e coloca o seu tamanho a 0.
FileAccess.Read	O ficheiro é aberto para leitura.
FileAccess.Write	O ficheiro é aberto para escrita.
FileAccess.ReadWrite	O ficheiro é aberto para leitura e escrita.

Exemplos

```
FileStream log = new FileStream("log.txt", FileMode.Append, FileAccess.Write);
```

```
FileStream log = new FileStream("log.txt", FileMode.Open, FileAccess.Read);
```

```
FileStream log = new FileStream("log.txt", FileMode.Append);
```

- Uma forma muito usual de utilização deste construtor é especificar apenas o modo de abertura. Neste caso o tipo de acesso é para leitura e escrita, o que cobre todas as situações possíveis de utilização.

Revisões

- Uma ***stream*** representa um fluxo de informação. A origem ou o destino da informação é escondido do programador;
- As *streams* podem ser combinadas oferecendo funcionalidades cada vez mais elaboradas. O programador chama os métodos da *stream* que se encontra mais acima da cadeia;
- **FileStream** representa a *stream* básica associada a um ficheiro;
- A forma mais frequente de criação desta *stream* é utilizando apenas o construtor onde se indica o nome do ficheiro e o modo de abertura;
- Muitas vezes também se utiliza o construtor de três parâmetros, indicando o modo de abertura e o tipo de acesso.

Ficheiros de texto

- Para ler dum ficheiro de texto, usa-se o objecto *StreamReader;*

- Para escrever para um ficheiro, usa-se o objecto *StreamWriter*;

```
FileStream fich = new FileStream("numeros.txt", FileMode.Create);
StreamWriter writer = new StreamWriter(fich);

writer.WriteLine("ciclo for de 0 a 100");

for(int i=0; i<101; i++)
        writer.WriteLine("{0}",i);

writer.Close();
```

NOTA: falta proteger este código com um bloco *try-catch*

Ficheiros de texto

- Para ler dum ficheiro de texto, o procedimento é similar ao anterior:

```
FileStream fich = new FileStream("numeros.txt", FileMode.Open);
StreamReader reader = new StreamReader(fich);

string linha;
do {
        linha = reader.ReadLine();
        if(linha != null)
                Console.WriteLine(linha);

} while(linha != null);

reader.Close();
```

- A classe *StreamReader* lê um ficheiro linha a linha.
- **NOTA: falta proteger este código com um bloco *try-catch***