



Programação e Sistemas de Informação

CURSO PROFISSIONAL TÉCNICO DE
GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

Tratamento de Ficheiros

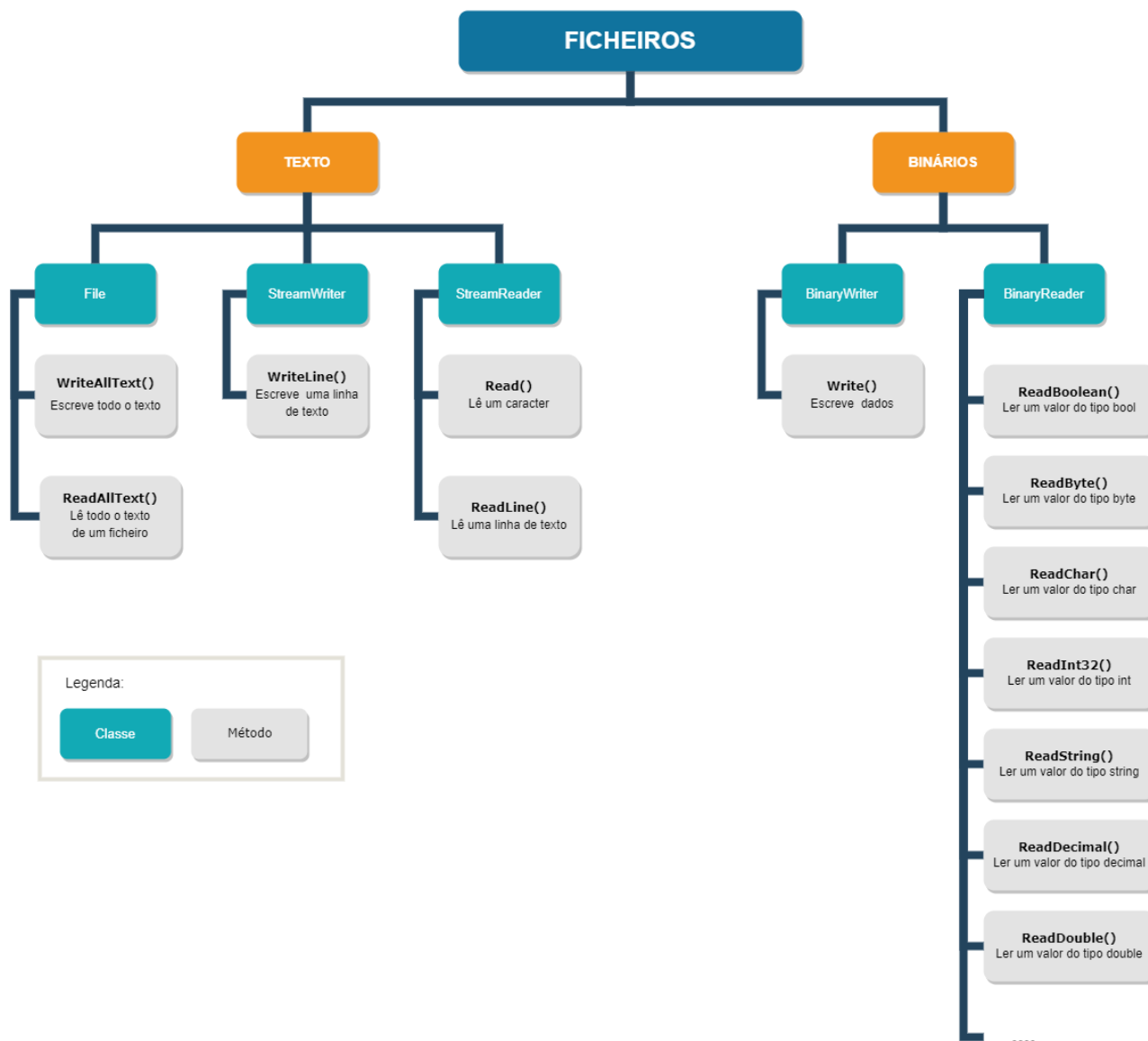
MÓDULO 7

Professor: João Martiniano

v1.0.1

Conteúdos abordados neste módulo

- Introdução aos ficheiros
- Operações com ficheiros de texto
 - Escrever dados
 - Ler dados
- Operações com ficheiros binários
 - Escrever dados
 - Ler dados



Introdução

- Em programação, os ficheiros têm um papel fundamental
- São uma forma de armazenamento de dados **permanente**, ao contrário da memória RAM que é volátil (ou seja, quando desligamos o computador, o conteúdo da memória desaparece)
- São utilizados para:
 - armazenar os dados de um programa (por exemplo: os níveis de um jogo, os dados dos pacientes numa clínica, os dados dos alunos de uma escola, etc.)
 - armazenar as configurações de um programa (por exemplo: as preferências do utilizador)

Tipos de ficheiros

- Existem dois tipos de ficheiros:
 - ficheiros binários
 - ficheiros de texto
- Cada um destes tipos de ficheiros armazena a informação de forma diferente
- E são utilizados com diferentes objetivos

Ficheiros binários

Ficheiros binários

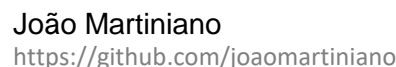
- Os ficheiros binários consistem em conjuntos de bytes
- Enquanto que um ficheiro de texto apenas contém texto
- Um ficheiro binário pode conter qualquer tipo de informação (texto, imagem, vídeo, etc.)
- Os ficheiros binários armazenam dados num formato que não é facilmente percecionável pelos seres humanos: contêm bytes
- Para visualizar corretamente o conteúdo de um ficheiro binário é habitual utilizar um *hex viewer*
- Os ficheiros binários necessitam de ser abertos com o programa correto, para que o seu conteúdo seja visualizado corretamente
- Por exemplo, um ficheiro **.mp4** é visualizado mais corretamente com o aplicativo VLC do que com o aplicativo Microsoft Word

Ficheiros binários

Ficheiros binários

- Exemplos de ficheiros binários:
 - ficheiros de imagem (.bmp, .gif, .png, .jpg, .webp, etc.)
 - ficheiros de documentos do Microsoft Office (.doc, .xls, .ppt, etc.)
 - ficheiros executáveis (.exe)
 - ficheiros de vídeo (.mp4, .mkv, .avi, etc.)
 - ficheiros comprimidos (.7z, .zip, .rar, .gz, etc.)
 - etc.

- Por exemplo, o seguinte ficheiro de imagem (.jpg) quando visualizado com o aplicativo Bloco de Notas, apresenta o seguinte conteúdo:



Ficheiros binários

- Abrindo este ficheiro num editor de código e utilizando visualização em formato hexadecimal, conseguimos ver o seu conteúdo de forma mais rigorosa:

```

1  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2  00000000: FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48 .X..JFIF....H
3  00000010: 00 48 00 00 FF E1 00 66 45 78 69 66 00 00 4D 4D H....a.fExif..MM
4  00000020: 00 2A 00 00 00 08 00 04 01 1A 00 05 00 00 00 01 *.
5  00000030: 00 00 00 3E 01 1B 00 05 00 00 00 01 00 00 00 46 ...>.....F
6  00000040: 01 28 00 03 00 00 00 01 00 03 00 00 01 31 00 02 .(.
7  00000050: 00 00 00 10 00 00 00 4E 00 00 00 00 00 00 6E BA .....N.....:
8  00000060: 00 00 03 E8 00 00 6E BA 00 00 03 E8 70 61 69 6E ...h..n:...hpain
9  00000070: 74 2E 6E 65 74 20 34 2E 31 2E 36 00 FF E2 0C 58 t.net.4.1.6..b.X
10 00000080: 49 43 43 5F 50 52 4F 46 49 4C 45 00 01 01 00 00 ICC_PROFILE....
11 00000090: 0C 48 4C 69 6E 6F 02 10 00 00 6D 6E 74 72 52 47 .Hlino...mnrRG
12 000000a0: 42 20 58 59 5A 20 07 CE 00 02 00 09 00 06 00 31 B.XYZ..N.....1
13 000000b0: 00 00 61 63 73 70 4D 53 46 54 00 00 00 00 49 45 ..acspMSFT...IE
14 000000c0: 43 20 73 52 47 42 00 00 00 00 00 00 00 00 00 00 C.sRGB.....
15 000000d0: 00 00 00 00 F6 D6 00 01 00 00 00 00 03 2D 48 50 ....vV.....S-HP
16 000000e0: 20 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
17 000000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
18 00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
19 00000110: 00 11 63 70 72 74 00 00 01 50 00 00 00 33 64 65 ..cprt...P...3de
20 00000120: 73 63 00 00 01 84 00 00 00 6C 77 74 70 74 00 00 sc.....lwtp..
21 00000130: 01 F0 00 00 00 14 62 68 70 74 00 00 02 04 00 00 .p....bkpt.....
22 00000140: 00 14 72 58 59 5A 00 00 02 18 00 00 00 14 67 58 ..rXYZ.....gX
23 00000150: 59 5A 00 00 02 2C 00 00 00 14 62 58 59 5A 00 00 YZ....bXYZ..
24 00000160: 02 40 00 00 00 14 64 6D 6E 64 00 00 02 54 00 00 .@....dmnd...T..
25 00000170: 00 70 64 6D 64 64 00 00 02 C4 00 00 00 88 76 75 .pdmd...D...vu
26 00000180: 65 64 00 00 03 4C 00 00 00 86 76 69 65 77 00 00 ed...L...view..
27 00000190: 03 D4 00 00 00 24 6C 75 6D 69 00 00 03 F8 00 00 .T...$lumi...x..
28 000001a0: 00 14 6D 65 61 73 00 00 04 0C 00 00 00 24 74 65 ..meas.....$te
29 000001b0: 63 68 00 00 04 30 00 00 00 0C 72 54 52 43 00 00 ch...0...rTRC..
30 000001c0: 04 3C 00 00 08 0C 67 54 52 43 00 00 04 3C 00 00 <....gTRC...<..
31 000001d0: 08 0C 62 54 52 43 00 00 04 3C 00 00 08 0C 74 65 ..bTRC...<....te
32 000001e0: 78 74 00 00 00 00 43 6F 70 79 72 69 67 68 74 20 xt...Copyright.
33 000001f0: 28 63 29 20 31 39 38 20 48 65 77 6C 65 74 74 (c).1998.Hewlett
34 00000200: 2D 50 61 63 68 61 72 64 20 43 6F 6D 70 61 6E 79 -Packard.Company
35 00000210: 00 00 64 65 73 63 00 00 00 00 00 00 12 73 52 ..desc.....sR
36 00000220: 47 42 20 49 45 43 36 31 39 36 36 2D 32 2E 31 00 GB.IEC61966-2.1.
  
```

Ficheiros binários

- Mas só abrindo o ficheiro num editor de imagem, é que o vemos corretamente:

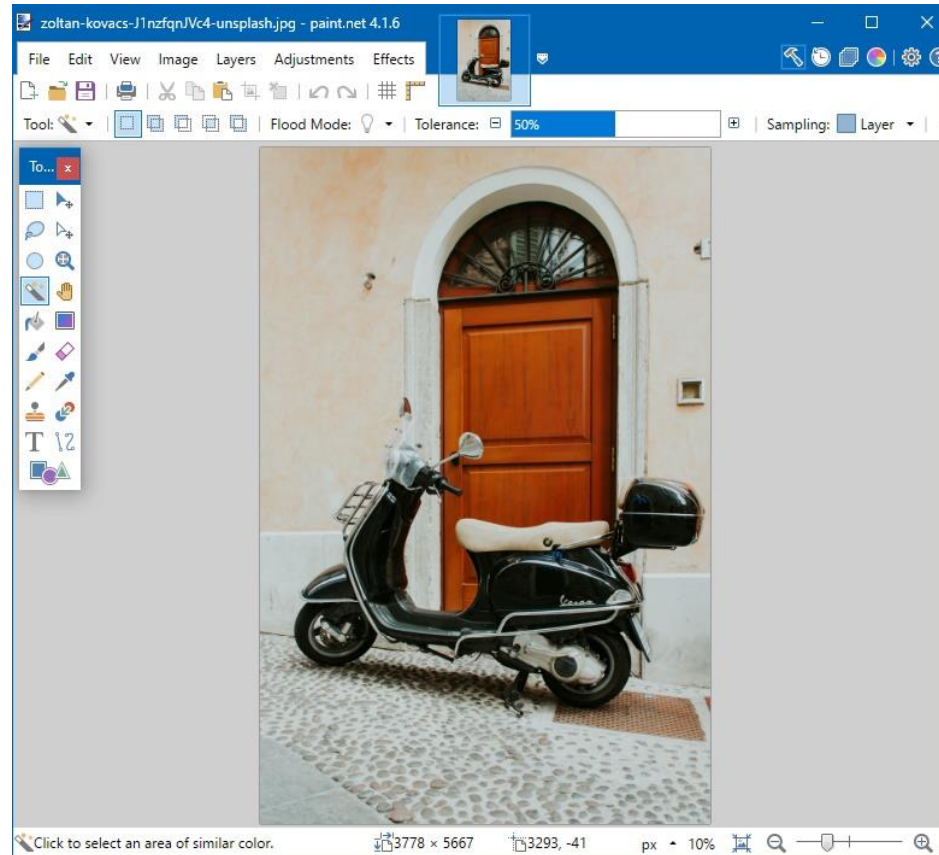


Imagem: Zoltan Kovacs (<https://unsplash.com/@kovacs21>)

Ficheiros de texto

- Os ficheiros de texto armazenam texto
- Consequentemente o conteúdo é de tipo simples
- O conteúdo pode ser visualizado diretamente embora possam existir determinados caracteres com uma função especial (para um editor de texto)
- Ficheiros de texto criados num sistema, podem aparecer um tanto ou quando "desformatados" quando abertos diretamente noutro sistema
- Nestas situações poderá ser necessário efetuar algum tipo de conversão

Ficheiros de texto

Atividade 1

- Abra o ficheiro [adidas-logo.svg](#) no programa **Inkscape** ou similar, e visualize a imagem vetorial (logotipo da Adidas):



- Feche o ficheiro e abra-o num editor de código (Visual Studio, Notepad++, etc.) ou texto (bloco de notas, etc.)

Ficheiros de texto

Atividade 1 (continuação)

- Localize o elemento `<text>` e mude o conteúdo de `adidas` para `adidas é fixe`
- Analise os atributos do elemento `<text>` e tente mudar a cor da letra para `#b88100`
- Grave o ficheiro e abra-o novamente no aplicativo **Inkscape**
- O resultado final deverá ser semelhante à seguinte imagem:



O namespace `System.IO`

- O namespace `System.IO` contém funcionalidades e recursos para trabalhar com ficheiros e diretorias
- Utilizando as classes deste namespace é possível criar/editar/eliminar ficheiros e diretorias
- Para trabalhar com **drives/diretorias**, utilizam-se as seguintes classes:

Classe	Descrição
<code>DriveInfo</code>	Informações sobre drives
<code>Directory</code>	Criar, eliminar, copiar, mover e renomear diretorias (métodos estáticos)
<code>DirectoryInfo</code>	Criar, eliminar, copiar, mover e renomear diretorias

O namespace `System.IO`

- Para trabalhar com **ficheiros**, utilizam-se as seguintes classes:

Classe	Descrição
<code>File</code>	Criar, eliminar, copiar, mover e renomear um ficheiro (métodos estáticos)
<code>FileInfo</code>	Criar, eliminar, copiar, mover e renomear ficheiros
<code>BinaryReader</code>	Ler dados binários
<code>BinaryWriter</code>	Escrever dados binários
<code>StreamReader</code>	Ler ficheiros de texto
<code>StreamWriter</code>	Escrever ficheiros de texto

Streams

- Ao escrever programas que utilizam ficheiros, o programador é confrontado com o conceito de *stream*
- Basicamente uma *stream* é uma fonte de dados, na qual é possível:
 - ler dados
 - escrever dados
 - percorrer a stream
- A fonte de dados pode ser um ficheiro, um dispositivo de *input/output*, etc.
- Como estas fontes de dados têm diferentes características e diferentes capacidades, as *streams* são uma forma de abstração
- Ou seja:
 - fornecem ao programador uma forma genérica de implementar as três operações (ler dados, escrever dados, percorrer)
 - o programador não se preocupa com a forma concreta como são implementadas as operações

OPERAÇÕES COM FICHEIROS DE TEXTO

Operações com ficheiros de texto

- Existem várias formas de trabalhar com ficheiros de texto
- A .NET Framework proporciona várias formas de atingir o mesmo resultado
- Neste documento são abordadas as seguintes classes e métodos:
 - **File**:
 - escrever dados utilizando o método `WriteAllText()`
 - ler dados utilizando o método `ReadAllText()`
 - **StreamWriter**: escrever dados para um ficheiro de texto
 - método `WriteLine()`
 - **StreamReader**: ler dados a partir de um ficheiro de texto
 - método `Read()`
 - método `ReadLine()`

Operações com ficheiros de texto

- Antes de começar é importante saber que:
 - é sempre necessário importar o namespace `System.IO`

```
using System.IO;
```

- especificar a localização e nome de ficheiros utilizando o caracter `@`

```
@ "ficheiro1.txt"
```

- os ficheiros, se não for especificada uma localização, são criados na diretoria `bin\Debug` dos programas

OPERAÇÕES COM FICHEIROS DE TEXTO: ESCREVER DADOS

O método `File.WriteAllText()`

- O método `File.WriteAllText()` é bastante conveniente para situações simples
- Este método...
 1. Cria um novo ficheiro de texto (se já existir um ficheiro com o nome especificado, este é substituído pelo novo ficheiro)
 2. Escreve uma string para o ficheiro
 3. Fecha o ficheiro

- Sintaxe:

```
void WriteAllText(string path, string conteúdo)
```

- Em que:

`path` A localização e nome do ficheiro

`conteúdo` A string a escrever no ficheiro

O método `File.WriteAllText()`

Exemplo: Criar dois ficheiros e escrever texto em cada um

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

Importante!

```
namespace Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            File.WriteAllText(@"texto1.txt", "The quick brown fox jumped over the lazy dog.");

            string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";
            File.WriteAllText(@"texto2.txt", livros);
        }
    }
}
```

O método `File.WriteAllText()`

Exemplo: Criar dois ficheiros e escrever texto em cada um

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

namespace Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            File.WriteAllText(@"texto1.txt", "The quick brown fox jumped over the lazy dog.");

            string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";
            File.WriteAllText(@"texto2.txt", livros);
        }
    }
}
```



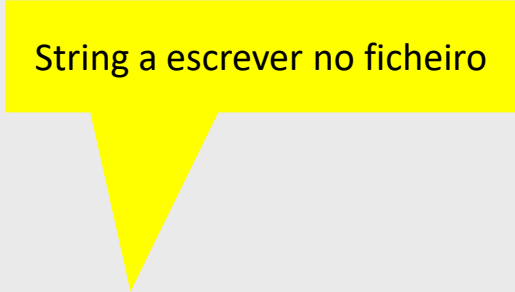
O método `File.WriteAllText()`

Exemplo: Criar dois ficheiros e escrever texto em cada um

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

namespace Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            File.WriteAllText(@"texto1.txt", "The quick brown fox jumped over the lazy dog.");

            string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";
            File.WriteAllText(@"texto2.txt", livros);
        }
    }
}
```



String a escrever no ficheiro

O método `File.WriteAllText()`

Exemplo: Criar dois ficheiros e escrever texto em cada um

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

namespace Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            File.WriteAllText(@"texto1.txt", "The quick brown fox jumped over the lazy dog.");

            string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";

            File.WriteAllText(@"texto2.txt", livros);
        }
    }
}
```

String a escrever no 2º
ficheiro

Criar o ficheiro `texto2.txt`

O método `File.WriteAllText()`

Exemplo: Criar dois ficheiros e escrever texto em cada um

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

namespace Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            File.WriteAllText(@"texto1.txt", "The quick brown fox jumped over the lazy dog.");

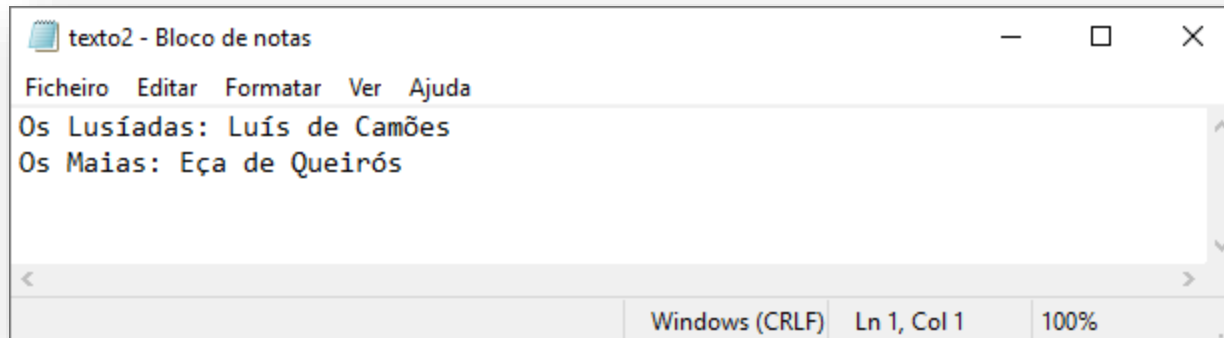
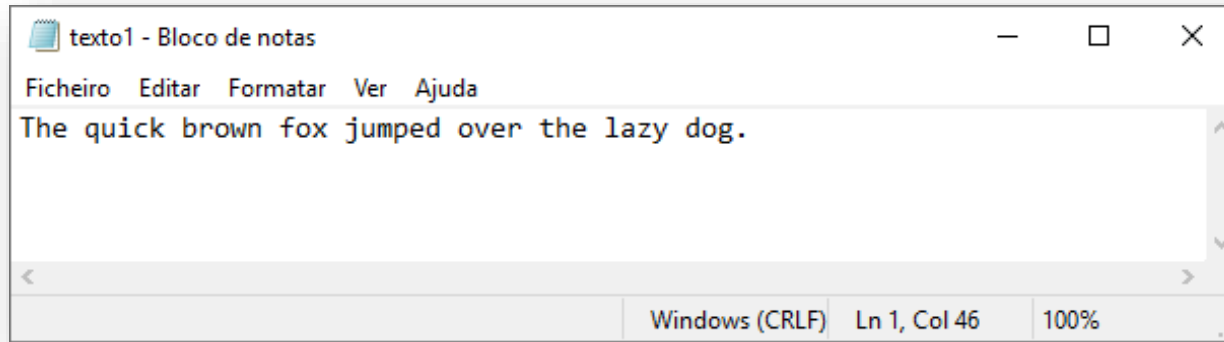
            string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";
            File.WriteAllText(@"texto2.txt", livros);
        }
    }
}
```



Inserir uma quebra de linha

O método `File.WriteAllText()`

- Resultado:



Tratamento de erros

- Nesta fase é importante abordar a importância de efetuar tratamento de erros
- Quando se trabalha com ficheiros, podem acontecer vários e inesperados erros, tais como:
 - o ficheiro pode estar danificado
 - o ficheiro pode não existir (nome incorreto, pode ter sido apagado, etc.)
 - a localização do ficheiro pode ter sido mal especificada
 - o ficheiro pode ter sido mudado de local
 - o dispositivo onde o ficheiro se encontra pode estar inacessível ou danificado
 - etc.
- É por isso que deve sempre haver tratamento de erros, para lidar com o que possa acontecer
- Neste módulo iremos fazer tratamento simplificado de erros, utilizando **exceções**

Tratamento de erros

- Resumidamente, as exceções funcionam da seguinte forma:
 - tenta-se efetuar uma operação
 - se ocorrer um erro, é gerada uma exceção
 - se o programa tiver tratamento de exceções o programador pode incluir código para lidar com a exceção gerada
- Sintaxe (muito simplificada e incompleta):

```
try
{
    // operação que pode dar origem a um erro
}
catch
{
    // código para tratar o erro
}
```

- (nota: as exceções serão lecionadas num módulo posterior)

Tratamento de erros

- O seguinte código exemplifica a utilização de exceções
- Para tal é modificado o exemplo analisado anteriormente, introduzindo propositadamente um erro: tentar criar os ficheiros num dispositivo inexistente
(verifique que no computador onde está a executar o exemplo, não existe nenhum dispositivo com as letras especificadas no exemplo)

```
try
{
    File.WriteAllText(@"W:\texto1.txt", "The quick brown fox jumped over the lazy dog.");
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro \"texto1.txt\".");
}

string livros = "Os Lusíadas: Luís de Camões" + Environment.NewLine + "Os Maias: Eça de Queirós";

try
{
    File.WriteAllText(@"Z:\texto2.txt", livros);
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro \"texto2.txt\".");
}
```

Não existe

Não existe

Tratamento de erros

- Neste documento, por razões de espaço, o código dos exemplos nem sempre irá incluir tratamento de erros
- **No entanto, deveremos sempre incluir um bloco `try...catch` sempre que tentarmos executar o código dos exemplos, bem como ao resolver as fichas deste módulo**

A classe `StreamWriter`

- A classe `StreamWriter` permite escrever dados para um ficheiro de texto
- Apesar de conter vários métodos para esta operação, iremos apenas utilizar o método `WriteLine()`
- Este método escreve, num ficheiro, uma string seguida de uma quebra de linha

- Sintaxe:

```
void WriteLine(string conteúdo)
```

- Em que:

`conteúdo` A string a escrever no ficheiro

A classe `StreamWriter`

Exemplo: Criar um ficheiro de texto e escrever uma string no ficheiro

```
try
{
    using (StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt"))
    {
        ficheiro.WriteLine("Hello World!");
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar/escrever no ficheiro.");
}
```

A classe `StreamWriter`

Exemplo: Criar um ficheiro de texto e escrever uma string no ficheiro

```
try
{
    using (StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt"))
    {
        ficheiro.WriteLine("Hello World!");
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar/escrever no ficheiro.");
}
```

Variável do tipo `StreamWriter`

A classe `StreamWriter`

Exemplo: Criar um ficheiro de texto e escrever uma string no ficheiro

```
try
{
    using (StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt"))
    {
        ficheiro.WriteLine("Hello World!");
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar/escrever no ficheiro.");
}
```

Nome do ficheiro

A classe `StreamWriter`

Exemplo: Criar um ficheiro de texto e escrever uma string no ficheiro

```
try
{
    using (StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt"))
    {
        ficheiro.WriteLine("Hello World!");
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar/escrever no ficheiro.");
}
```

Escrever a string "Hello World!" no ficheiro

A instrução `using`

- Sempre que trabalhamos com ficheiros, é necessário, no final, fechá-los
- Para garantir que os dados são corretamente escritos nos ficheiros
- No entanto, utilizando a instrução `using` não é necessário fazê-lo explicitamente
- Esta instrução encarrega-se de fechar os ficheiros com os quais trabalhamos
- Utilizando a instrução `using` (estilo de programação recomendado):

```
using (StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt"))  
{  
    ficheiro.WriteLine("Hello World!");  
}
```

- Sem utilizar a instrução `using`:

```
StreamWriter ficheiro = new StreamWriter(@"ficheiro.txt");  
ficheiro.WriteLine("Hello World!");  
ficheiro.Flush();  
ficheiro.Close();
```

A classe `StreamWriter`

Exercício 1:

- Complete os espaços no código para:
 - criar um novo ficheiro intitulado `escola.txt`
 - escrever a string `"Escola Secundária Avelar Brotero"` no ficheiro

```
using System;
using System.____;

namespace Exercicio1
{
    class Program
    {
        static void Main(string[] args)
        {
            using (_____ ficheiro = new StreamWriter(@"_____"))
            {
                ficheiro.WriteLine("_____");
            }
        }
    }
}
```

A classe `StreamWriter`

Exercício 1: Resolução

- Complete os espaços no código para:
 - criar um novo ficheiro intitulado `escola.txt`
 - escrever a string `"Escola Secundária Avelar Brotero"` no ficheiro

```
using System;
using System.IO;

namespace Exercicio1
{
    class Program
    {
        static void Main(string[] args)
        {
            using (StreamWriter ficheiro = new StreamWriter(@"escola.txt"))
            {
                ficheiro.WriteLine("Escola Secundária Avelar Brotero");
            }
        }
    }
}
```

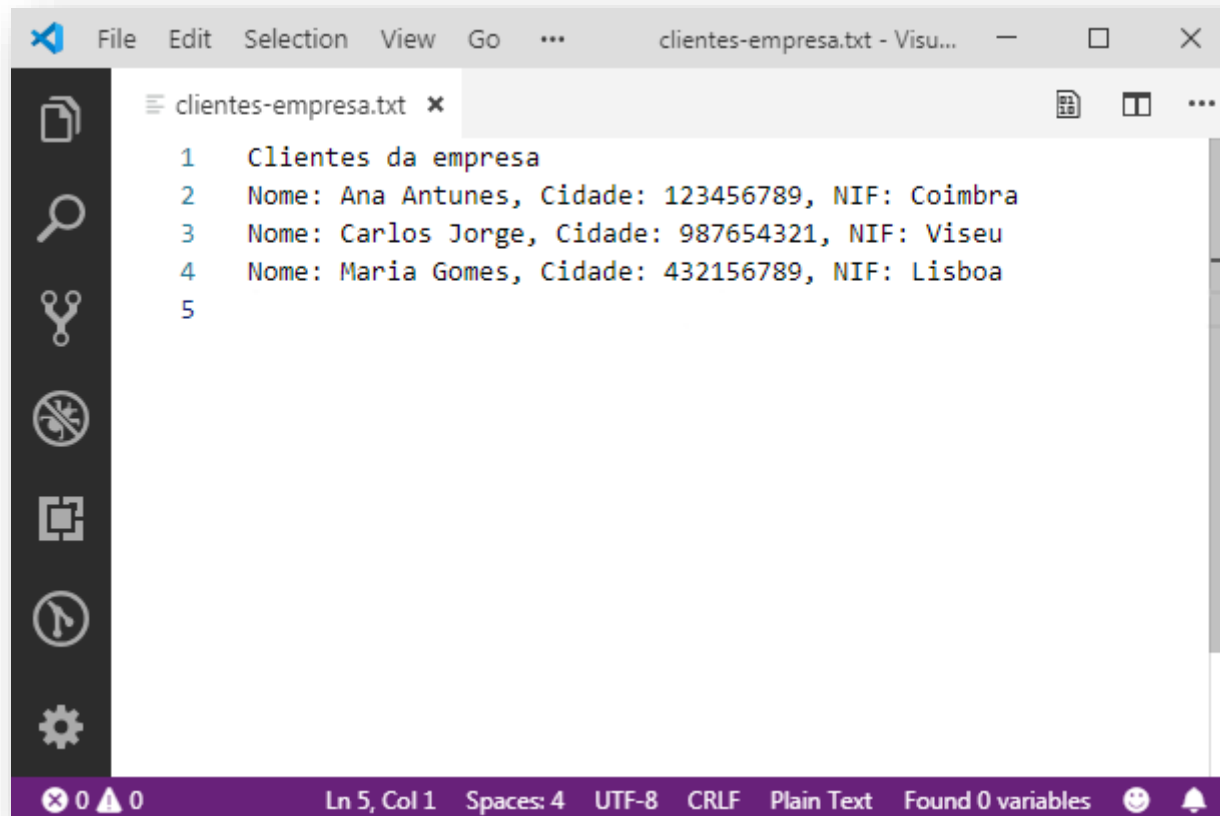
A classe StreamWriter

Exemplo: Declarar variáveis com dados de clientes e escrever o conteúdo no ficheiro `clientes-empresa.txt`

```
string cliente1 = "Ana Antunes";  
int cliente1Nif = 123456789;  
string cliente1Cidade = "Coimbra";  
  
string cliente2 = "Carlos Jorge";  
int cliente2Nif = 987654321;  
string cliente2Cidade = "Viseu";  
  
string cliente3 = "Maria Gomes";  
int cliente3Nif = 432156789;  
string cliente3Cidade = "Lisboa";  
  
using (StreamWriter ficheiro = new StreamWriter(@"clientes-empresa.txt"))  
{  
    ficheiro.WriteLine("Clientes da empresa");  
    ficheiro.WriteLine($"Nome: {cliente1}, Cidade: {cliente1Nif}, NIF: {cliente1Cidade}");  
    ficheiro.WriteLine($"Nome: {cliente2}, Cidade: {cliente2Nif}, NIF: {cliente2Cidade}");  
    ficheiro.WriteLine($"Nome: {cliente3}, Cidade: {cliente3Nif}, NIF: {cliente3Cidade}");  
}
```


A classe `StreamWriter`

Exemplo: Conteúdo do ficheiro



```
File Edit Selection View Go ... clientes-empresa.txt - Visu...  
clientes-empresa.txt x  
1 Clientes da empresa  
2 Nome: Ana Antunes, Cidade: 123456789, NIF: Coimbra  
3 Nome: Carlos Jorge, Cidade: 987654321, NIF: Viseu  
4 Nome: Maria Gomes, Cidade: 432156789, NIF: Lisboa  
5  
Ln 5, Col 1 Spaces: 4 UTF-8 CRLF Plain Text Found 0 variables
```

OPERAÇÕES COM FICHEIROS DE TEXTO:

LER DADOS

O método `File.ReadAllText()`

- O método `File.ReadAllText()` é bastante conveniente para situações simples
- Este método...
 1. Abre um ficheiro de texto
 2. Lê todo o texto do ficheiro
 3. Fecha o ficheiro

- Sintaxe:

```
string ReadAllText(string path)
```

- Em que:

`path`

A localização e nome do ficheiro

O método `File.ReadAllText()`

Exemplo: Ler o conteúdo de um ficheiro e mostrá-lo na consola

```
string texto = string.Empty;

try
{
    texto = File.ReadAllText(@"texto1.txt");
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}

Console.WriteLine("Foi lido o seguinte texto, a partir do ficheiro:");
Console.WriteLine(texto);
```

O método `File.ReadAllText()`

Exemplo: Ler o conteúdo de um ficheiro e mostrá-lo na consola

```
string texto = string.Empty;  
  
try  
{  
    texto = File.ReadAllText(@"texto1.txt");  
}  
catch  
{  
    Console.WriteLine("Não foi possível tentar utilizar o ficheiro.");  
}  
  
Console.WriteLine("Foi lido o seguinte texto, a partir do ficheiro:");  
Console.WriteLine(texto);
```

Abrir o ficheiro `texto1.txt`

O conteúdo do ficheiro é colocado na variável `texto`

O método `File.ReadAllText()`

Exemplo: Ler o conteúdo de um ficheiro e mostrá-lo na consola

```
string texto = string.Empty;

try
{
    texto = File.ReadAllText(@"texto1.txt");
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}

Console.WriteLine("Foi lido o seguinte texto, a partir do ficheiro:");
Console.WriteLine(texto);
```

Mostrar na consola o
conteúdo de `texto1.txt`

A classe `StreamReader`

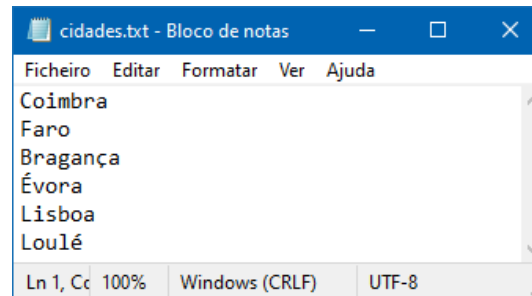
- A classe `StreamReader` permite ler dados a partir de um ficheiro de texto
- Apesar de conter vários métodos para esta operação, iremos apenas utilizar os métodos `Read()` e `ReadLine()`
 - `Read()`: lê um carácter a partir de um ficheiro e retorna o carácter em formato `Int32` (terá depois de ser convertido para `char`)
 - `ReadLine()`: lê uma linha, de um ficheiro de texto, e retorna uma string com os dados lidos
- Sintaxe:
`int Read()`
`string ReadLine()`

A classe `StreamReader`

- Ao ler dados de um ficheiro é importante testar se o final do mesmo foi atingido
- Caso tenha sido atingido o final de um ficheiro e se tente continuar a ler o seu conteúdo, é gerado um erro
- Existem diferentes formas de testar
- Por exemplo:
 - utilizar o método `Peek()`
 - verificar se a string retornada pelo método `ReadLine()` é igual a `null`

A classe `StreamReader`: Método `ReadLine()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, linha a linha



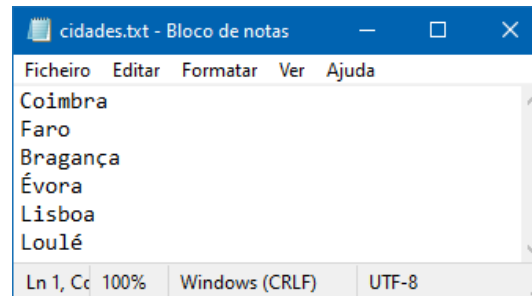
Abrir o ficheiro `idades.txt`

```
string linha;

using (StreamReader ficheiro = new StreamReader(@"idades.txt"))
{
    while ((linha = ficheiro.ReadLine()) != null)
    {
        Console.WriteLine(linha);
    }
}
```

A classe `StreamReader`: Método `ReadLine()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, linha a linha

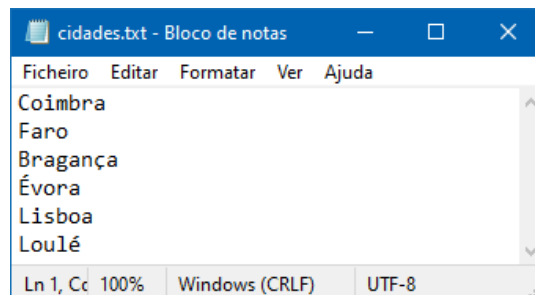


```
string linha;  
  
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))  
{  
    while ((linha = ficheiro.ReadLine()) != null)  
    {  
        Console.WriteLine(linha);  
    }  
}
```

Ler uma linha do ficheiro e atribuir à variável `linha`

A classe `StreamReader`: Método `ReadLine()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, linha a linha



```
string linha;

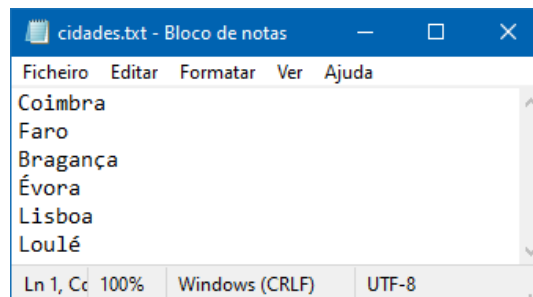
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))
{
    while ((linha = ficheiro.ReadLine()) != null)
    {
        Console.WriteLine(linha);
    }
}
```

Verificar que não foi atingido o fim do ficheiro:

- se `linha == null` foi atingido o fim
- se `linha != null` não foi atingido o fim

A classe `StreamReader`: Método `ReadLine()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, linha a linha



```
string linha;  
  
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))  
{  
    while ((linha = ficheiro.ReadLine()) != null)  
    {  
        Console.WriteLine(linha);  
    }  
}
```

Ler as linhas do ficheiro enquanto
não tiver sido atingido o final
do mesmo

A classe `StreamReader`: Método `Read()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, caracter a caracter

Abrir o ficheiro `idades.txt`

```
char c;  
  
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))  
{  
    while (ficheiro.Peek() >= 0)  
    {  
        c = (char)ficheiro.Read();  
        Console.WriteLine(c);  
    }  
}
```

A classe `StreamReader`: Método `Read()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, caracter a caracter

```
char c;  
  
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))  
{  
    while (ficheiro.Peek() >= 0)  
    {  
        c = (char)ficheiro.Read();  
        Console.WriteLine(c);  
    }  
}
```

Verificar se foi atingido o fim do ficheiro:

- se `Peek() == -1` foi atingido o fim
- se `Peek() >= 0` não foi atingido o fim

A classe `StreamReader`: Método `Read()`

- Exemplo: Ler e mostrar o conteúdo do ficheiro `idades.txt`, caracter a caracter

```
char c;  
  
using (StreamReader ficheiro = new StreamReader(@"idades.txt"))  
{  
    while (ficheiro.Peek() >= 0)  
    {  
        c = (char)ficheiro.Read();  
        Console.WriteLine(c);  
    }  
}
```

Ler um caracter e converter
para `char`

OPERAÇÕES COM FICHEIROS BINÁRIOS

Operações com ficheiros binários

- Os ficheiros binários permitem uma maior liberdade de armazenamento dos dados
- É possível não só armazenar texto mas dados de outros tipos (inteiros, reais, etc.)
- O programador tem maior controlo sobre como os dados são armazenados

Operações com ficheiros binários

- Iremos utilizar duas classes para trabalhar com ficheiros binários:
 - **BinaryWriter**: escrever dados em ficheiros binários
 - contém o método **Write()** para escrever os dados
 - **BinaryReader**: ler dados a partir de ficheiros binários
 - são utilizados vários métodos de acordo com os dados que se pretendem ler:
 - **ReadBoolean()**: ler um valor do tipo **bool**
 - **ReadByte()**: ler um valor do tipo **byte**
 - **ReadChar()**: ler um valor do tipo **char**
 - **ReadInt32()**: ler um valor do tipo **int**
 - **ReadString()**: ler um valor do tipo **string**
 - **ReadDecimal()**: ler um valor do tipo **decimal**
 - **ReadDouble()**: ler um valor do tipo **double**
 - etc.

Operações com ficheiros binários

- Para demonstrar como trabalhar com ficheiros binários iremos analisar o seguinte exemplo:
 - um programa que utiliza a struct **Aluno** para armazenar dados de alunos
 - os dados são armazenados numa **List** do tipo **Aluno** (**List<Aluno>**)
- Este exemplo demonstra como:
 - escrever os dados dos alunos num ficheiro binário
 - ler os dados dos alunos a partir de um ficheiro binário

Operações com ficheiros binários

- Vamos começar por criar a struct `Aluno` para armazenar os dados dos alunos:

```
public struct Aluno
{
    public string Turma;
    public string Nome;
    public int Numero;
    public bool PrimeiraMatricula;

    public Aluno(string turma, string nome, int numero, bool primeiraMatricula)
    {
        Turma = turma;
        Nome = nome;
        Numero = numero;
        PrimeiraMatricula = primeiraMatricula;
    }
}
```

Operações com ficheiros binários

- De seguida, no método `main()`, vamos declarar e inicializar uma lista com 4 elementos:

```
List<Aluno> alunos = new List<Aluno>();  
  
alunos.Add(new Aluno("11PSI4", "Alberto", 1, false));  
alunos.Add(new Aluno("11PSI4", "Carla", 2, true));  
alunos.Add(new Aluno("11PSI4", "Mónica", 3, false));  
alunos.Add(new Aluno("10PM", "André", 1, false));
```

Operações com ficheiros binários

- O passo seguinte consiste em:
 - **criar um novo ficheiro binário**
 - percorrer a lista de alunos
 - escrever os dados de cada aluno no ficheiro

```
try
{
    using (BinaryWriter writer = new BinaryWriter(File.Open(@"dados-alunos.dat", FileMode.Create)))
    {
        foreach (Aluno a in alunos)
        {
            writer.Write(a.Turma);
            writer.Write(a.Nome);
            writer.Write(a.Numero);
            writer.Write(a.PrimeiraMatricula);
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro.");
}
```

Criar um novo ficheiro

Operações com ficheiros binários

- O passo seguinte consiste em:
 - **criar um novo ficheiro binário**
 - percorrer a lista de alunos
 - escrever os dados de cada aluno no ficheiro

```
try
{
    using (BinaryWriter writer = new BinaryWriter(File.Open(@"dados-alunos.dat", FileMode.Create)))
    {
        foreach (Aluno a in alunos)
        {
            writer.Write(a.Turma);
            writer.Write(a.Nome);
            writer.Write(a.Numero);
            writer.Write(a.PrimeiraMatricula);
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro.");
}
```

Instanciar a classe **BinaryWriter** com base no ficheiro criado

Operações com ficheiros binários

- O passo seguinte consiste em:
 - criar um novo ficheiro binário
 - **percorrer a lista de alunos**
 - **escrever os dados de cada aluno no ficheiro**

```
try
{
    using (BinaryWriter writer = new BinaryWriter(File.Open(@"dados-alunos.dat", FileMode.Create)))
    {
        foreach (Aluno a in alunos)
        {
            writer.Write(a.Turma);
            writer.Write(a.Nome);
            writer.Write(a.Numero);
            writer.Write(a.PrimeiraMatricula);
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro.");
}
```

Percorrer a lista de alunos

Escrever os dados de cada aluno no ficheiro

Operações com ficheiros binários

- O passo seguinte consiste em:
 - criar um novo ficheiro binário
 - **percorrer a lista de alunos**
 - **escrever os dados de cada aluno no ficheiro**

```
try
{
    using (BinaryWriter writer = new BinaryWriter(File.Open(@"dados-alunos.dat", FileMode.Create)))
    {
        foreach (Aluno a in alunos)
        {
            writer.Write(a.Turma);
            writer.Write(a.Nome);
            writer.Write(a.Numero);
            writer.Write(a.PrimeiraMatricula);
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro.");
}
```

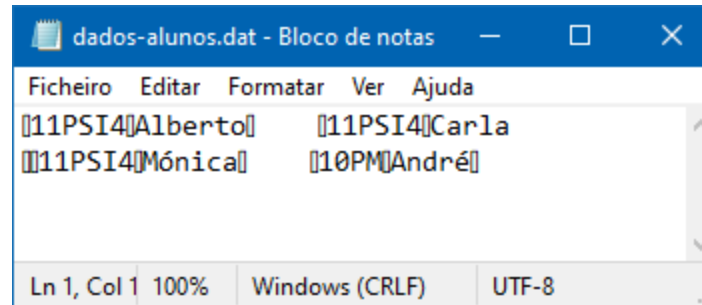
Tenha em atenção que o ficheiro
irá conter uma sequência de:

string
string
int
bool

(para cada aluno)

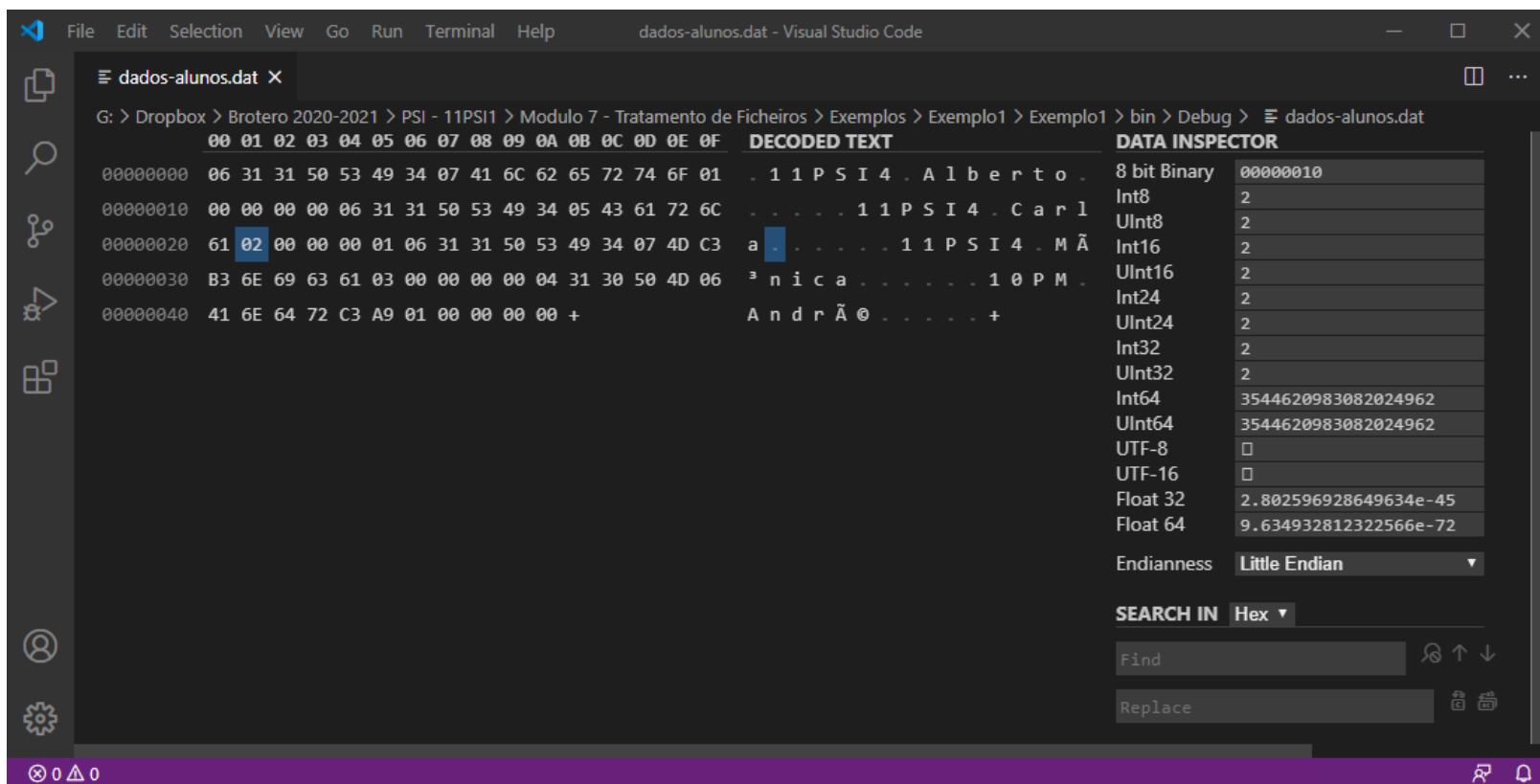
Operações com ficheiros binários

- Eis o conteúdo do ficheiro quando aberto diretamente no bloco de notas:



Operações com ficheiros binários

- O mesmo ficheiro, quando visualizado através de um *hex viewer*:



- Lembre-se que esta é a maneira mais correta de visualizar o conteúdo de um ficheiro binário

Operações com ficheiros binários

- Para ler os dados do ficheiro binário:
 - começamos por declarar as variáveis que irão armazenar os dados

```
List<Aluno> alunos = new List<Aluno>();  
  
// Guardar temporariamente os dados lidos diretamente do ficheiro  
string turma, nome;  
int numero;  
bool primeiraMatricula;
```

Operações com ficheiros binários

- De seguida, abrimos o ficheiro, lemos os dados e adicionamos à lista de alunos

```
try
{
    using (BinaryReader reader = new BinaryReader(File.Open(@"dados-alunos.dat",
    FileMode.Open)))
    {
        while (reader.PeekChar() >= 0)
        {
            turma = reader.ReadString();
            nome = reader.ReadString();
            numero = reader.ReadInt32();
            primeiraMatricula = reader.ReadBoolean();

            alunos.Add(new Aluno(turma, nome, numero, primeiraMatricula));
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}
```

Abrir o ficheiro, em modo de leitura

Operações com ficheiros binários

- De seguida, abrimos o ficheiro, lemos os dados e adicionamos à lista de alunos

```
try
{
    using (BinaryReader reader = new BinaryReader(File.Open(@"dados-alunos.dat",
    FileMode.Open)))
    {
        while (reader.PeekChar() >= 0)
        {
            turma = reader.ReadString();
            nome = reader.ReadString();
            numero = reader.ReadInt32();
            primeiraMatricula = reader.ReadBoolean();

            alunos.Add(new Aluno(turma, nome, numero, primeiraMatricula));
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}
```

Instanciar a classe `BinaryReader` com base no ficheiro aberto

Operações com ficheiros binários

- De seguida, abrimos o ficheiro, lemos os dados e adicionamos à lista de alunos

```
try
{
    using (BinaryReader reader = new BinaryReader(File.Open(
        FileMode.Open)))
    {
        while (reader.PeekChar() >= 0)
        {
            turma = reader.ReadString();
            nome = reader.ReadString();
            numero = reader.ReadInt32();
            primeiraMatricula = reader.ReadBoolean();

            alunos.Add(new Aluno(turma, nome, numero, primeiraMatricula));
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}
```

Note que para ler os dados, são utilizados diferentes métodos:

- `ReadString()`
- `ReadInt32()`
- `ReadBoolean()`

Operações com ficheiros binários

- De seguida, abrimos o ficheiro, lemos os dados e adicionamos à lista de alunos

```
try
{
    using (BinaryReader reader = new BinaryReader(File.Open("dados-alunos.dat",
        FileMode.Open)))
    {
        while (reader.PeekChar() >= 0)
        {
            turma = reader.ReadString();
            nome = reader.ReadString();
            numero = reader.ReadInt32();
            primeiraMatricula = reader.ReadBoolean();

            alunos.Add(new Aluno(turma, nome, numero, primeiraMatricula));
        }
    }
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar utilizar o ficheiro.");
}
```

Percorrer o ficheiro até atingir o fim

Ler os dados

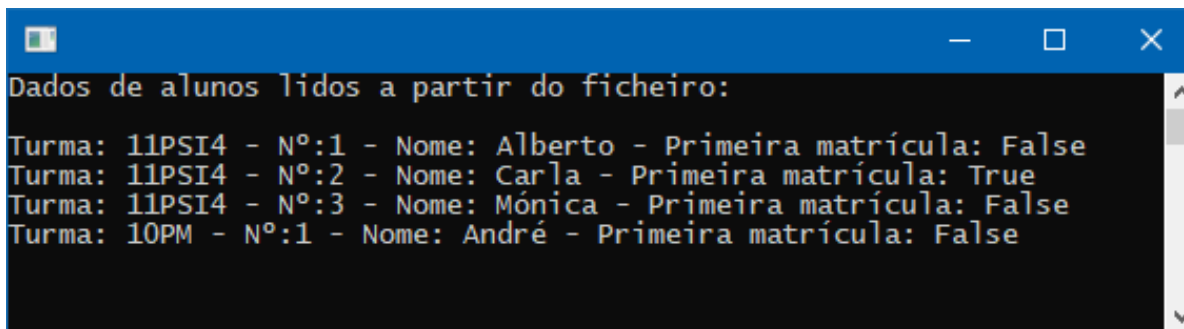
Adicionar um novo aluno à lista

Operações com ficheiros binários

- Com os dados na lista de alunos, basta percorrê-la para mostrar os dados de cada aluno na consola:

```
Console.WriteLine("Dados de alunos lidos a partir do ficheiro:\n");  
  
foreach (Aluno a in alunos)  
{  
    Console.WriteLine($"Turma: {a.Turma} - Nº:{a.Numero} - Nome: {a.Nome} -  
    Primeira matrícula: {a.PrimeiraMatricula}");  
}
```

- Resultado:



INTERAÇÃO COM O FILESYSTEM

Interação com o *filesystem*

- A .NET Framework disponibiliza vários recursos para que o programador possa interagir com o *filesystem* (sistema de ficheiros) de um computador
- Neste documento são abordados apenas alguns desses recursos, nomeadamente:
 - obter dados de um ficheiro
 - obter dados de uma pasta
 - obter dados de um disco
 - eliminar, copiar, mover e verificar se um ficheiro existe

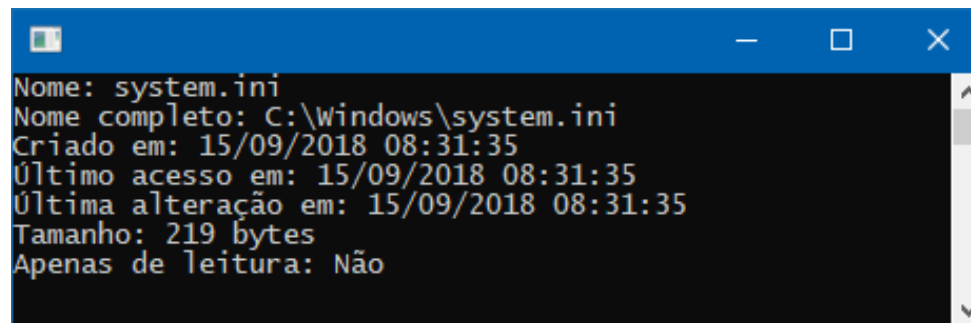
Interação com o *filesystem*: Detalhes de um ficheiro

- Exemplo: utilizar a classe `FileInfo` para obter dados de um ficheiro

```
FileInfo ficheiro = new FileInfo(@"C:\Windows\system.ini");

Console.WriteLine($"Nome: {ficheiro.Name}");
Console.WriteLine($"Nome completo: {ficheiro.FullName}");
Console.WriteLine($"Criado em: {ficheiro.CreationTime}");
Console.WriteLine($"Último acesso em: {ficheiro.LastAccessTime}");
Console.WriteLine($"Última alteração em: {ficheiro.LastWriteTime}");
Console.WriteLine($"Tamanho: {ficheiro.Length} bytes");
Console.WriteLine($"Apenas de leitura: {ficheiro.IsReadOnly} ? "Sim"
: "Não");
```

- Resultado:



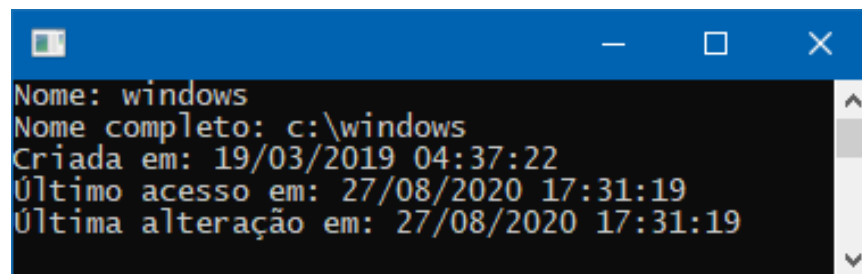
```
Nome: system.ini
Nome completo: C:\Windows\system.ini
Criado em: 15/09/2018 08:31:35
Último acesso em: 15/09/2018 08:31:35
Última alteração em: 15/09/2018 08:31:35
Tamanho: 219 bytes
Apenas de leitura: Não
```

Interação com o *filesystem*: Detalhes de uma pasta

- Exemplo: utilizar a classe `DirectoryInfo` para obter dados de uma pasta

```
DirectoryInfo pasta = new DirectoryInfo(@"c:\windows");  
  
Console.WriteLine("Nome: {0}", pasta.Name);  
Console.WriteLine("Nome completo: {0}", pasta.FullName);  
Console.WriteLine("Criada em: {0}", pasta.CreationTime);  
Console.WriteLine("Último acesso em: {0}", pasta.LastAccessTime);  
Console.WriteLine("Última alteração em: {0}", pasta.LastWriteTime);
```

- Resultado:



```
Nome: windows  
Nome completo: c:\windows  
Criada em: 19/03/2019 04:37:22  
Último acesso em: 27/08/2020 17:31:19  
Última alteração em: 27/08/2020 17:31:19
```

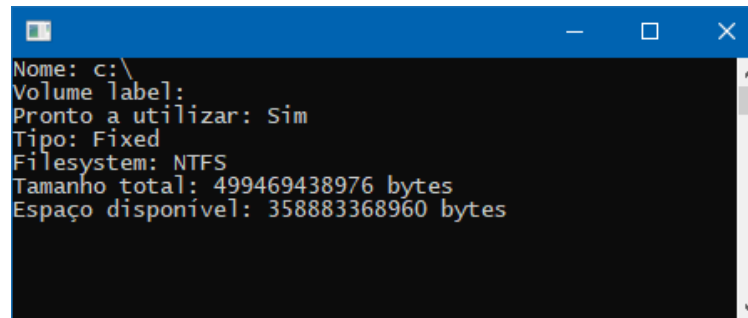
Interação com o *filesystem*: Detalhes de um disco

- Exemplo: utilizar a classe `DriveInfo` para obter dados de um disco

```
DriveInfo disco = new DriveInfo("c");

Console.WriteLine("Nome: {0}", disco.Name);
Console.WriteLine("Volume label: {0}", disco.VolumeLabel);
Console.WriteLine("Pronto a utilizar: {0}", (disco.IsReady) ? "Sim" :
    "Não");
Console.WriteLine("Tipo: {0}", disco.DriveType);
Console.WriteLine("Filesystem: {0}", disco.DriveFormat);
Console.WriteLine("Tamanho total: {0} bytes", disco.TotalSize);
Console.WriteLine("Espaço disponível: {0} bytes", disco.TotalFreeSpace);
```

- Resultado:



```
Nome: c:\
Volume label:
Pronto a utilizar: Sim
Tipo: Fixed
Filesystem: NTFS
Tamanho total: 499469438976 bytes
Espaço disponível: 358883368960 bytes
```

Interação com o *filesystem*: Ficheiros

- A classe `File` contém vários recursos para trabalhar com ficheiros
- Alguns métodos disponibilizados por esta classe:
 - `File.Copy()`: copia um ficheiro
 - `File.Delete()`: apaga um ficheiro
 - `File.Exists()`: verifica se um ficheiro existe
 - `File.Move()`: move um ficheiro

O método `File.Copy()`

- Sintaxe:

```
void ReadAllText(string nomeAntigo, string novoNome)
```

- Em que:

<code>nomeAntigo</code>	A localização e o nome do ficheiro a copiar
<code>novoNome</code>	A localização e novo nome do ficheiro (não pode ser o nome de um ficheiro já existente, no local de destino)

O método `File.Copy()`

- Exemplo:

```
string origem = @"C:\texto.txt";
string destino = @"D:\texto.txt";

try
{
    if (File.Exists(origem))
    {
        File.Copy(origem, destino);

        Console.WriteLine($"Ficheiro \"{origem}\" copiado com sucesso para
        \"{destino}\"");
    }
}
catch
{
    Console.WriteLine($"Não foi possível copiar o ficheiro \"{origem}\"
    para \"{destino}\"");
}
```

O método `File.Delete()`

- Sintaxe:

```
void Delete(string path)
```

- Em que:

`path` A localização e o nome do ficheiro a apagar

- Exemplo:

```
string ficheiro = @"C:\texto.txt";

try
{
    File.Delete(ficheiro);
    Console.WriteLine("Ficheiro \"{ficheiro}\" eliminado com sucesso");
}
catch
{
    Console.WriteLine("Não foi possível eliminar o ficheiro  
 \"{ficheiro}\"");
}
```

O método `File.Exists()`

- Sintaxe:

`bool Exists(string path)`

- Em que:

`path` A localização e o nome do ficheiro a determinar se existe

- Retorna:

`true` Se o ficheiro existe na localização especificada

`false` Caso contrário

O método `File.Exists()`

- Exemplo:

```
string ficheiro = @"C:\texto.txt";

if (File.Exists(ficheiro))
{
    Console.WriteLine("O ficheiro \"{ficheiro}\" existe");
}
else
{
    Console.WriteLine("O ficheiro \"{ficheiro}\" não existe");
}
```

O método `File.Move()`

- Sintaxe:

```
void Move(string ficheiroOrigem, string ficheiroDestino)
```

- Em que:

`ficheiroOrigem` A localização e o nome do ficheiro a mover

`ficheiroDestino` A localização e (opcionalmente) novo nome para onde o ficheiro é movido

O método `File.Move()`

- Exemplo:

```
string origem = @"C:\temp\texto.txt";
string destino = @"C:\temp2\texto.txt";

try
{
    // Determinar que o ficheiro a mover existe
    if (File.Exists(origem))
    {
        // Determinar que o ficheiro não existe no destino
        if (!File.Exists(destino))
        {
            File.Move(origem, destino);
            Console.WriteLine("Ficheiro movido com sucesso.");
        }
        else
        {
            Console.WriteLine("O ficheiro não foi movido porque já existe no destino");
        }
    }
}
catch
{
    Console.WriteLine("Não foi possível mover o ficheiro.");
}
```