Fundamentos de C#

Armazenamento em Arquivos

Softblue
cursos online

Tópicos Abordados

- Arquivos e diretórios
- Drives
- Leitura e escrita de dados
  - *FileStream*
  - *BinaryReader* e *BinaryWriter*
  - *StreamReader* e *StreamWriter*

Dados em Arquivos

- Muitas aplicações precisam ser capazes de manipular arquivos
- Arquivos podem ser dos tipos
  - Binário
  - Texto
- Operações em arquivos são operações de I/O (input-output)

Arquivos e Diretórios

- Existem 4 classes que podem ser utilizadas para realizar operações em arquivos e diretórios
  - Arquivos
    - *File* / *FileInfo*
  - Diretórios
    - *Directory* / *DirectoryInfo*
- As classes possuem basicamente as mesmas funcionalidades
  - *File* / *Directory* só possuem métodos estáticos
  - *FileInfo* / *DirectoryInfo* precisam ser instanciadas
- Estas classes pertencem ao namespace *System.IO*

Arquivos

- Criando um novo arquivo

```
FileInfo f = new FileInfo(@"C:\Temp\file.txt");
f.Create();
```

```
File.Create(@"C:\Temp\file.txt");
```

- Copiando um arquivo para outro local

```
FileInfo f = new FileInfo(@"C:\Temp\file.txt");
f.CopyTo(@"C:\Temp\file2.txt");
```

```
File.Copy(@"C:\Temp\file.txt", @"C:\Temp\file2.txt");
```

- Verificando a existência de um arquivo

```
FileInfo f = new FileInfo(@"C:\Temp\file.txt");
bool b = f.Exists;
```

```
bool b = File.Exists(@"C:\Temp\file.txt");
```

Arquivos

- Usando um objeto *FileInfo*, é possível acessar algumas properties

Property	Descrição
Extension	Extensão do arquivo
FullName	Caminho completo do arquivo
Name	Nome do arquivo (sem diretórios)
Length	Tamanho do arquivo
CreationTime	Data/hora de criação do arquivo
LastAccessTime	Data/hora do último acesso ao arquivo
LastWriteTime	Data/hora da última gravação no arquivo
Directory	Diretório onde o arquivo está contido

Diretórios

- Criando um novo diretório

```
DirectoryInfo d = new DirectoryInfo(@"C:\Temp\Curso");
d.Create();
```

```
Directory.CreateDirectory(@"C:\Temp\Curso");
```

- Movendo um diretório para outro local

```
DirectoryInfo d = new DirectoryInfo(@"C:\Temp\Curso");
d.MoveTo(@"C:\Temp\Curso2");
```

```
Directory.Move(@"C:\Temp\Curso", @"C:\Temp\Curso2");
```

- Obtendo os arquivos do diretório

```
DirectoryInfo d = new DirectoryInfo(@"C:\Temp\Curso");
FileInfo[] files = d.GetFiles("*.txt");
```

```
string[] files = Directory.GetFiles(@"C:\Temp\Curso", "*.txt");
```

Diretórios

- Usando um objeto *DirectoryInfo*, é possível acessar algumas properties

Property	Descrição
Exists	Checa existência do diretório
FullName	Caminho completo do diretório
Name	Nome do diretório
CreationTime	Data/hora de criação do diretório
LastAccessTime	Data/hora do último acesso ao diretório
LastWriteTime	Data/hora da última gravação no diretório
Parent	Diretório anterior na hierarquia

A Classe *Path*

- Possui métodos estáticos com alguns utilitários que facilitam o trabalho com caminhos de arquivos e diretórios

Método	Descrição
Combine()	Combina duas ou mais strings em um caminho de arquivo ou diretório
GetDirectoryName()	Obtém o nome do diretório a partir de um caminho
GetFileNameWithoutExtension()	Retorna o nome do arquivo sem a extensão
ChangeExtension()	Muda a extensão de um arquivo

A Classe *DriveInfo*

- Permite obter informações sobre os drives disponíveis

```
DriveInfo drive = new DriveInfo(@"C:\");
```

```
DriveInfo[] drives = DriveInfo.GetDrives();
```

Property	Descrição
Name	Nome do drive
AvailableFreeSpace	Espaço disponível no drive
IsReady	Indica se o drive está pronto
DriveFormat	Formato do sistema de arquivos do drive
TotalSize	Tamanho total do drive
VolumeLabel	Label associada ao drive

Classe *File* e a Leitura de Dados

- A classe *File* tem alguns métodos estáticos que podem ser usados para ler dados de arquivos texto ou binários
  - Ler todos os bytes do arquivo

```
byte[] bytes = File.ReadAllBytes(@"C:\Temp\file.bin");
```

  - Ler todos as linhas do arquivo

```
string[] lines = File.ReadAllLines(@"C:\Temp\file.txt");
```

  - Ler o arquivo como uma *string*

```
string text = File.ReadAllText(@"C:\Temp\file.txt");
```

Classe *File* e a Escrita de Dados

- A classe *File* tem alguns métodos estáticos que podem ser usados para escrever dados em arquivos texto ou binários
  - Escrever bytes no arquivo

```
byte[] bytes = ...;
File.WriteAllBytes(@"C:\Temp\file.bin", bytes);
```

  - Escrever linhas no arquivo

```
string[] lines = ...;
File.WriteAllLines(@"C:\Temp\file.txt", lines);
```

  - Escrever uma *string* no arquivo

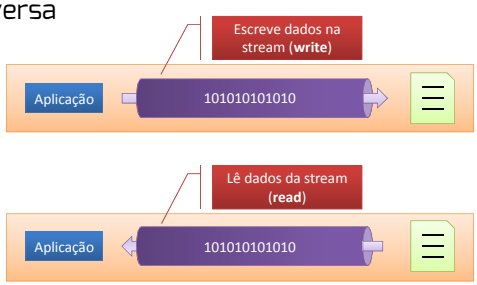
```
string text = ...;
File.WriteAllText(@"C:\Temp\file.txt", text);
```

Streams

- Stream é um conceito importante quando o assunto é I/O
- Uma stream é um canal por onde trafegam bytes, de uma origem a um destino

Streams em Arquivos

- Streams podem ser usadas para transferir dados da aplicação para um arquivo e vice-versa



A Classe *FileStream*

- Utilizada para ler ou escrever dados em um arquivo
- Usada normalmente com arquivos binários

```
FileStream fs = new FileStream(
    @"C:\Temp\file.bin", FileMode.Create, FileAccess.Write);
```

Arquivo | Modo | Acesso

Enumeration	Valores
FileMode	*Append, Create, CreateNew, Open, OpenOrCreate, Truncate*
FileAccess	*Read, ReadWrite, Write*

Obtendo um Objeto *FileStream*

- Um *FileStream* pode ser criado com o uso de um dos construtores da classe
- É possível criar um *FileStream* a partir de um objeto *FileInfo*

```
FileInfo file = new FileInfo(@"C:\Temp\file.bin");
```

```
FileStream fs = file.OpenRead();
```
Arquivo para leitura

```
FileStream fs = file.OpenWrite();
```
Arquivo para escrita

```
FileStream fs = file.Open(FileMode.OpenOrCreate, FileAccess.Read);
```
Permite especificar o modo e o acesso

O Método *Close()*

- Depois de utilizar o arquivo, é importante chamar o método *Close()*, para liberar os recursos utilizados

```
fs.Close();
```

```
fs.Dispose();
```

```
FileStream fs = file.OpenRead();
try
{
    //...
}
finally
{
    fs.Close();
}
```
Garante a chamada ao *Close()* mesmo na presença de uma exceção

```
using (FileStream fs = file.OpenRead())
{
    //...
}
```
Chama o *Dispose()* ao sair do bloco (tendo exceção ou não)

Leitura e Escrita em um *FileStream*

- Um objeto *FileStream* suporta a leitura e a escrita de dados
  - Dependendo do modo em que o arquivo foi aberto
- As operações de **ReadByte()** e **WriteByte()** podem ser utilizadas

```
byte b = fs.ReadByte();
```
Lê 1 byte

```
fs.WriteByte(b);
```
Escreve 1 byte

Toda operação de leitura e escrita avança para o próxima posição de leitura ou escrita

Leitura e Escrita em um *FileStream*

- É mais comum utilizar os métodos **Read()** e **Write()**
  - Ler ou escrever mais de um byte
  - Utilizam um buffer de bytes

```
byte[] bytes = ...;
int l = fs.Read(bytes, 0, bytes.Length);
```

Lê os dados para dentro de *bytes*

Retorna o número de bytes lidos

```
byte[] bytes = ...;
fs.Write(bytes, 0, bytes.Length);
```

Escreve os dados armazenados em *bytes*

Readers e *Writers*

- C# possui algumas classes com os sufixos *Reader* e *Writer*
- Estas classes não representam streams, mas podem se integrar a streams e fornecer funcionalidades de mais alto nível
- Classes importantes
  - **BinaryReader**
  - **BinaryWriter**
  - **StreamReader**
  - **StreamWriter**

Classes *BinaryReader* e *BinaryWriter*

- Trabalham com a leitura e escrita de tipos de dados como *int*, *long*, *double*, etc.

```
BinaryWriter w = new BinaryWriter(fs);
w.Write(x);
w.Write(y);
```

A stream é fornecida no construtor

```
BinaryReader r = new BinaryReader(fs);
int x = r.ReadInt32();
bool y = r.ReadBoolean();
```

Variações do no formato *Read*()*

Classes *StreamReader* e *StreamWriter*

- Utilizada para trabalhar com caracteres
  - Melhor opção para arquivos texto

```
StreamReader r = new StreamReader(fs);
StreamWriter w = new StreamWriter(fs);
```

A stream é fornecida no construtor

```
StreamReader r = File.OpenText(@"C:\Temp\file.txt");
StreamWriter w = File.CreateText(@"C:\Temp\file.txt");
```

A classe *File* pode fornecer os objetos

Lendo Caracteres com *StreamReader*

- Ler uma linha do arquivo

```
string line = r.ReadLine();
```

- Ler até o final do arquivo

```
string text = r.ReadToEnd();
```

- Ler um caractere

```
char c = (char) r.Read();
```

Escrevendo Caracteres com *StreamWriter*

- Escrever uma *string* no arquivo

```
w.Write("Texto qualquer");
```

- Escrever uma *string* e quebrar a linha

```
w.WriteLine("Texto qualquer");
```

- Escrever um caractere

```
w.Write(c);
```


Softblue