



Colégio
Pedro II

PROGRAMAÇÃO O.O.

(C#)



Entrada e Saída de Arquivos
Professor: João Luiz Lagôas



Streams

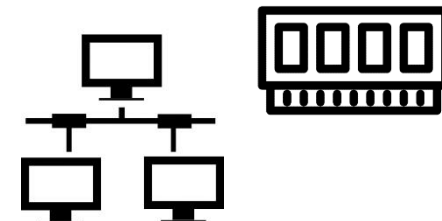


- Uma *stream* é como o Framework .NET transfere dados do seu programa para entidades externas (ou vice-versa), isto é, através de fluxo de dados.

stream



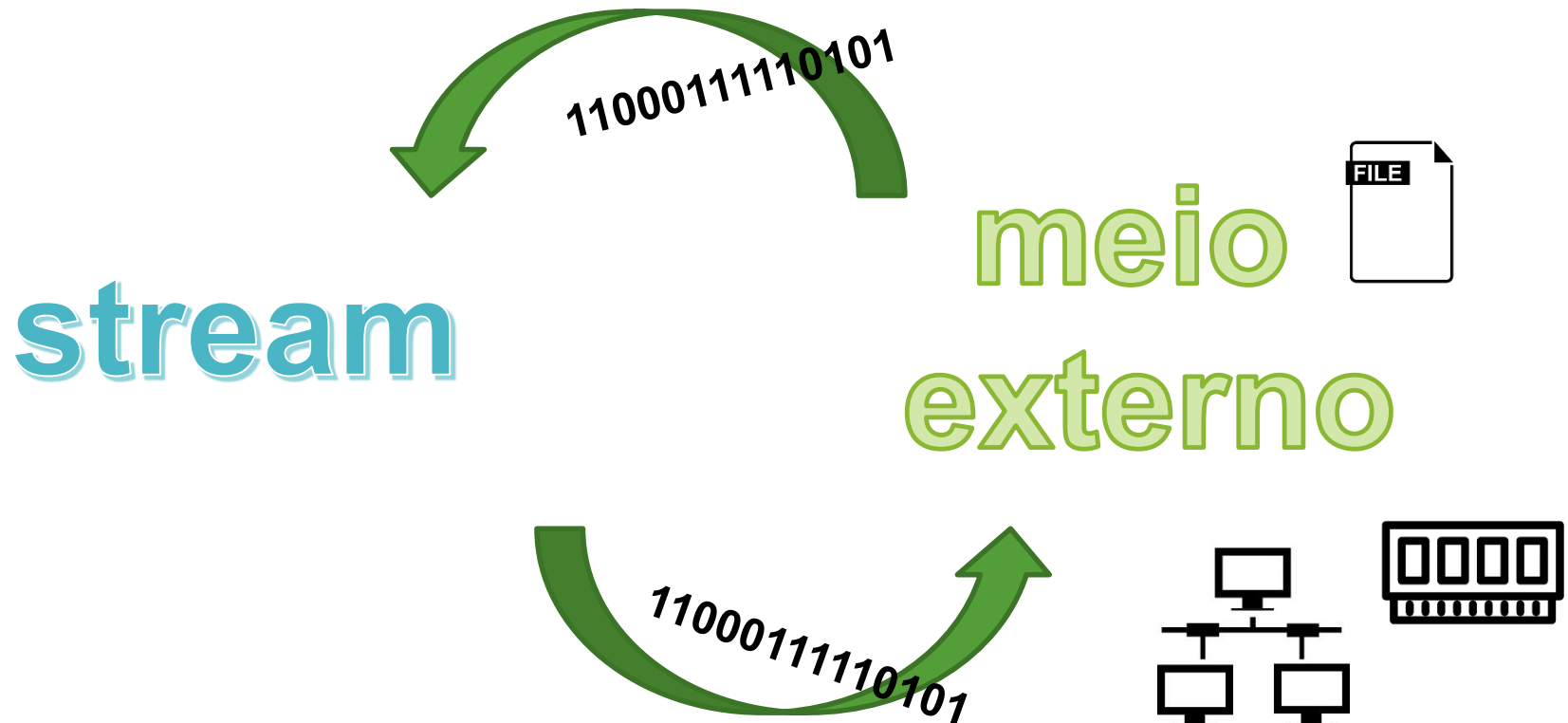
meio
externo



Streams



- A transferência dos dados sempre acontece em termos de **bytes** ao invés de string, int, float, double, etc.



StreamWriter e StreamReaders



- Vários detalhes da utilização de Streams podem ser simplificados utilizando as classes **StreamWriter** e **StreamReader**.
- Essas classes fazem todas essas coisas sem se preocupar com detalhes muito técnicos.

StreamWriter

ESCRITOR



LEITOR

StreamReader

StreamWriter



- Uma StreamWriter é utilizada para escrever em um arquivo.
- Em seu construtor, basta especificar onde o arquivo a ser trabalhado se encontra.
- O método WriteLine() é capaz de escrever em um arquivo apenas recebendo uma string como parâmetro.
- O método Write() também pode ser utilizado e não pula uma linha.

StreamWriter

Funcionamento



Colégio
Pedro II

Use o construtor de StreamWriter para abrir ou criar um arquivo.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
                        Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        StreamWriter escritor = new StreamWriter(path);

        escritor.WriteLine("Olá {0}!", nome);
        escritor.Write("Tenha um ótimo dia!");

        escritor.Close();
    }
}
```

StreamWriter

Funcionamento



Colégio
Pedro II

Use o método Write() e WriteLine() para escrever no arquivo que você especificou.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
                    Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        StreamWriter escritor = new StreamWriter(path);

        escritor.WriteLine("Olá {0}!", nome);
        escritor.Write("Tenha um ótimo dia!");

        escritor.Close();
    }
}
```



StreamWriter

Funcionamento



Colégio
Pedro II

Chame o método Close() para liberar o arquivo.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
                    Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        StreamWriter escritor = new StreamWriter(path);

        escritor.WriteLine("Olá {0}!", nome);
        escritor.Write("Tenha um ótimo dia!");

        escritor.Close();
    }
}
```


StreamReaders



- Uma `StreamReader` funciona exatamente igual a uma `StreamWriter`, mas em vez de escrever num arquivo, informa-se a ele o nome do arquivo que deve ser lido em seu construtor.
- O método `ReadLine()` retorna uma sequência de caracteres (string) com a próxima linha do arquivo.
- Você pode escrever um laço que leia linhas até que seu atributo `EndOfStream` seja `true` – é quando ele não tem mais linhas para ler:

StreamReader

Funcionamento



Colégio
Pedro II

Use o construtor de StreamReader para abrir o arquivo que deseja ler.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João Lagôas\Desktop\MyTest.txt";
        StreamReader leitor = new StreamReader(path);
        int i = 1;
        while(!leitor.EndOfStream)
        {
            string linhaDeTexto = leitor.ReadLine();
            Console.WriteLine("Linha {0}: {1}", i, linhaDeTexto);
            i++;
        }
        leitor.Close();
        Console.ReadLine();
    }
}
```

StreamReader

Funcionamento



Colégio
Pedro II

O atributo EndOfStream armazena true se o cursor estiver no fim do arquivo e false caso ainda haja informação para ler.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João Lagôas\Desktop\MyTest.txt";

        StreamReader leitor = new StreamReader(path);

        int i = 1;
        while(!leitor.EndOfStream)
        {
            string linhaDeTexto = leitor.ReadLine();
            Console.WriteLine("Linha {0}: {1}", i, linhaDeTexto);
            i++;
        }

        leitor.Close();

        Console.ReadLine();
    }
}
```

StreamReader

Funcionamento



Colégio
Pedro II

Use o método `ReadLine()` para recuperar uma linha de informação do arquivo e retorná-la como string.


```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João Lagôas\Desktop\MyTest.txt";

        StreamReader leitor = new StreamReader(path);

        int i = 1;
        while(!leitor.EndOfStream)
        {
            string linhaDeTexto = leitor.ReadLine();
            Console.WriteLine("Linha {0}: {1}", i, linhaDeTexto);
            i++;
        }

        leitor.Close();

        Console.ReadLine();
    }
}
```



StreamReader

Funcionamento



Colégio
Pedro II

Chame o método Close() para liberar o arquivo.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João Lagôas\Desktop\MyTest.txt";

        StreamReader leitor = new StreamReader(path);

        int i = 1;
        while(!leitor.EndOfStream)
        {
            string linhaDeTexto = leitor.ReadLine();
            Console.WriteLine("Linha {0}: {1}", i, linhaDeTexto);
            i++;
        }

        leitor.Close();

        Console.ReadLine();
    }
}
```

Comando `using`



- Fechar streams ao término de seu uso é uma tarefa importante. Falhas comuns enfrentadas pelos programadores quando lidam com arquivos são causadas por streams não fechadas corretamente.
- Quando você embute seu código stream dentro de um bloco **`using`**, ele automaticamente fecha suas streams para você.
- Ao final do bloco `using`, o C# automaticamente toma as providências necessárias para finalizar o uso do objeto de modo adequado.

Comando using



Antes

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
                        Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        StreamWriter escritor = new StreamWriter(path);

        escritor.WriteLine("Olá {0}!", nome);
        escritor.Write("Tenha um ótimo dia!");

        escritor.Close();
    }
}
```

Comando using



Depois

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
                        Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        using (StreamWriter escritor = new
                StreamWriter(path))
        {
            escritor.WriteLine("Olá {0}!", nome);
            escritor.Write("Tenha um ótimo dia!");
        }
    }
}
```


Comando using



Depois

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"C:\Users\João
            Lagôas\Desktop\MyTest.txt";

        string nome = "João";

        using (StreamWriter escritor = new
            StreamWriter(path))
        {
            escritor.WriteLine("Olá {0}!", nome);
            escritor.Write("Tenha um ótimo dia!");
        }
    }
}
```

O C# irá fechar o escritor ao fim desse bloco de execução

Comando using



- Você pode aninhar comandos using sem precisar adicionar mais pares de chaves.

```
class Program
{
    static void Main(string[] args)
    {
        string srcPath = @"C:\Users\João Lagôas\Desktop\origem.txt";
        string destPath = @"C:\Users\João Lagôas\Desktop\destino.txt";

        using (StreamWriter escritor = new StreamWriter(destPath))
        using (StreamReader leitor = new StreamReader(srcPath))
        {
            while (!leitor.EndOfStream)
            {
                string linha = leitor.ReadLine();
                escritor.WriteLine(linha);
            }
        }
    }
}
```



Comando using

- Você pode aninhar comandos using sem precisar adicionar mais pares de chaves.

```
class Program
{
    static void Main(string[] args)
    {
        string srcPath = @"C:\Users\João Lagôas\Desktop\origem.txt";
        string destPath = @"C:\Users\João Lagôas\Desktop\destino.txt";

        using (StreamWriter escritor = new StreamWriter(destPath))
        using (StreamReader leitor = new StreamReader(srcPath))
        {
            while (!leitor.EndOfStream)
            {
                string linha = leitor.ReadLine();
                escritor.WriteLine(linha);
            }
        }
    }
}
```

Quando a execução do programa encontra o fim do escopo do using, os objetos escritor e leitor serão fechados automaticamente.

Classes *File* e *Directory*

Ganhando intuição



- O .NET possui duas classes nativas com vários métodos estáticos para trabalhar com arquivos e pastas. A classe *File* fornece métodos para trabalhar com arquivos e a *Directory* permite lidar com diretórios. Escreva o que você acha que fazem essas linhas de código.

Classes *File* e *Directory*

Ganhando intuição



Código	O que o código faz
<pre>if(!Directory.Exists(@"C:\LP2")){ Directory.CreateDirectory(@"C:\LP2"); }</pre>	
<pre>if (Directory.Exists(@"C:\LP2")){ Directory.Delete(@"C:\LP2"); }</pre>	
<pre>File.Copy(@"C:\LP2\notas.txt", @"D:\LP2\nt.txt");</pre>	
<pre>DateTime myTime = Directory.GetCreationTime(@"C:\LP2\nota s.txt");</pre>	
<pre>File.Delete(@"C:\LP2\notas.txt");</pre>	
<pre>File.WriteAllText(@"C:\LP2\materia2C.txt", @"Programação O.O., muitos controles, encapsulamento, herança avançada, polimorfismo, acesso a arquivos");</pre>	

Classes *File* e *Directory*

Ganhando intuição



Código	O que o código faz
<pre>if(!Directory.Exists(@"C:\LP2")){ Directory.CreateDirectory(@"C:\LP2"); }</pre>	<p>Checa se a pasta LP2 existe. Se não, uma pasta LP2 é criada.</p>
<pre>if (Directory.Exists(@"C:\LP2")){ Directory.Delete(@"C:\LP2"); }</pre>	<p>Checa se a pasta LP2 existe. Se sim, ela é deletada.</p>
<pre>File.Copy(@"C:\LP2\notas.txt", @"D:\LP2\nt.txt");</pre>	<p>Copia o arquivo notas.txt para o arquivo nt.txt.</p>
<pre>DateTime myTime = Directory.GetCreationTime(@"C:\LP2\nota s.txt");</pre>	<p>Declara uma variável myTime e a atribui ao retorno do método. Note que esse método retorna um objeto do tipo DateTime.</p>
<pre>File.Delete(@"C:\LP2\notas.txt");</pre>	<p>Deleta o arquivo notas.txt.</p>
<pre>File.WriteAllText(@"C:\LP2\materia2C.txt", @"Programação O.O., muitos controles, encapsulamento, herança avançada, polimorfismo, acesso a arquivos");</pre>	<p>Cria um arquivo chamado materia2C.txt (se ele não existir) e escreve o conteúdo passado como segundo parâmetro.</p>

Classes *File* e *Directory*



- Assim como StreamWriter, a classe **File** cria, nos bastidores, streams para você trabalhar com arquivos. Você pode usar seus métodos para realizar ações mais comuns sem ter de criar FileStreams primeiro.
- Os objetos **Directory** permitem trabalhar com diretórios inteiros cheios de arquivos e pode-se usá-los para fazer alterações na estrutura de pastas facilmente.
- A **documentação** contendo todos os atributos/propriedades e métodos dessas classes pode ser encontrada em:
 - [https://msdn.microsoft.com/pt-br/library/system.io.file\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/system.io.file(v=vs.110).aspx)
 - [https://msdn.microsoft.com/pt-br/library/system.io.directory\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/system.io.directory(v=vs.110).aspx)

Classe *File*



- Coisas que você pode fazer com a classe File
 1. Descobrir se um arquivo existe: você pode checar para ver se um arquivo existe usando método **Exists()**.
 2. Ler e escrever no arquivo: você pode usar o método **OpenRead()** para acessar dados de um arquivo, ou o método **Create()** ou **OpenWrite()** para escrever nele.
 3. Concatenar texto em um arquivo: O método **AppendAllText()** permite anexar texto a um arquivo criado. Inclusive, ele cria o arquivo se este não estiver lá quando o método executar.
 4. Obter informações:
 - o método **GetLastAccessTime()** e **GetLastWriteTime()** retornam a data e hora do último acesso e alteração do arquivo.

Classe *Directory*



- Coisas que você pode fazer com a classe `Directory`
 1. Criar um novo diretório: Crie um diretório usando o método **`CreateDirectory()`**. Tudo a fazer é fornecer o caminho que o método fará todo o resto.
 2. Obter uma lista de arquivos no diretório: você pode criar uma matriz de arquivos em um diretório usando o método **`GetFiles()`**. Apenas informe ao método um diretório e ele se encarregará de tudo.
 3. Apagar um diretório é bem simples também. Use o método **`Delete()`**.

