



PROGRAMAÇÃO O.O. (C#)

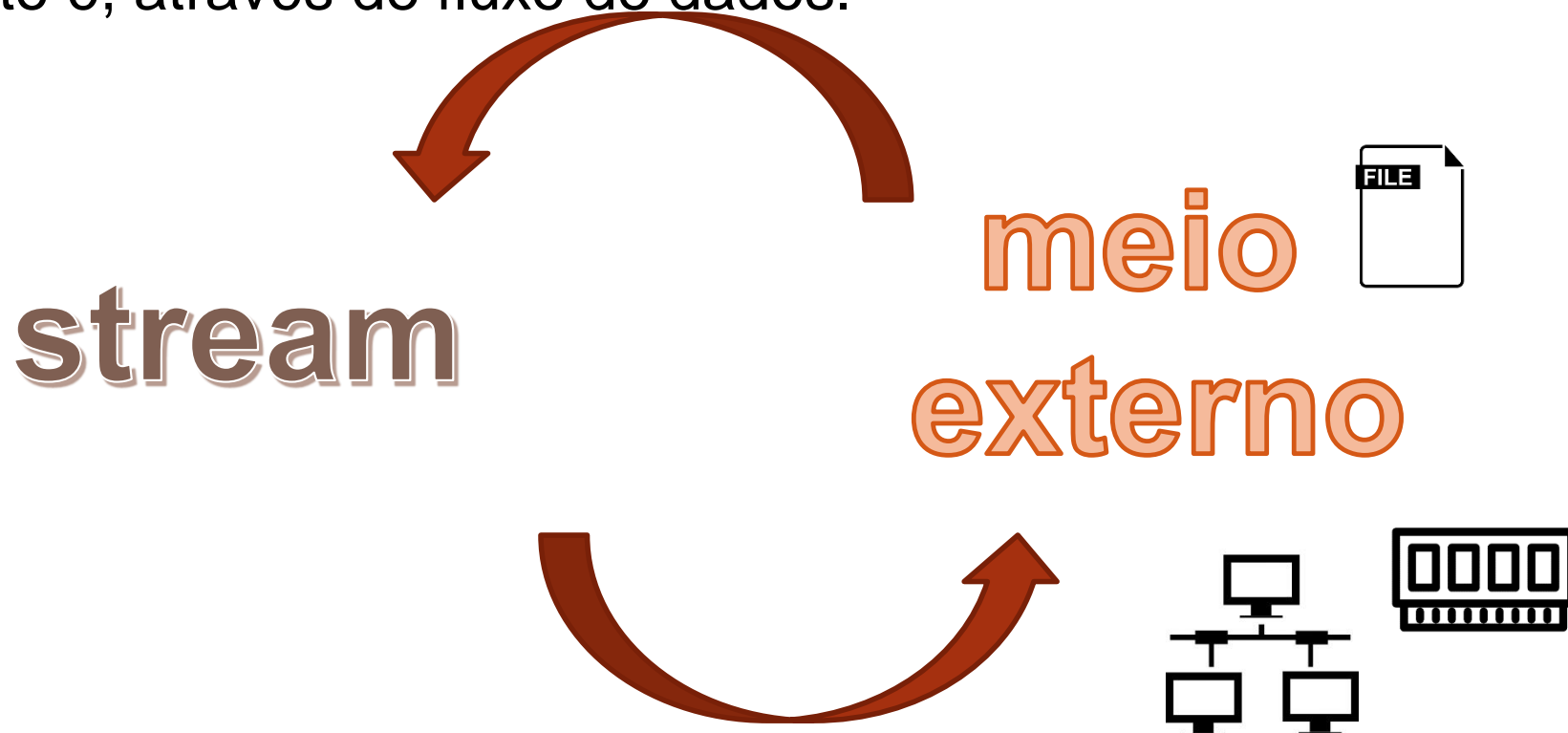
{ Entrada e Saída de Arquivos
Professor: João Luiz Lagôas

}

Streams



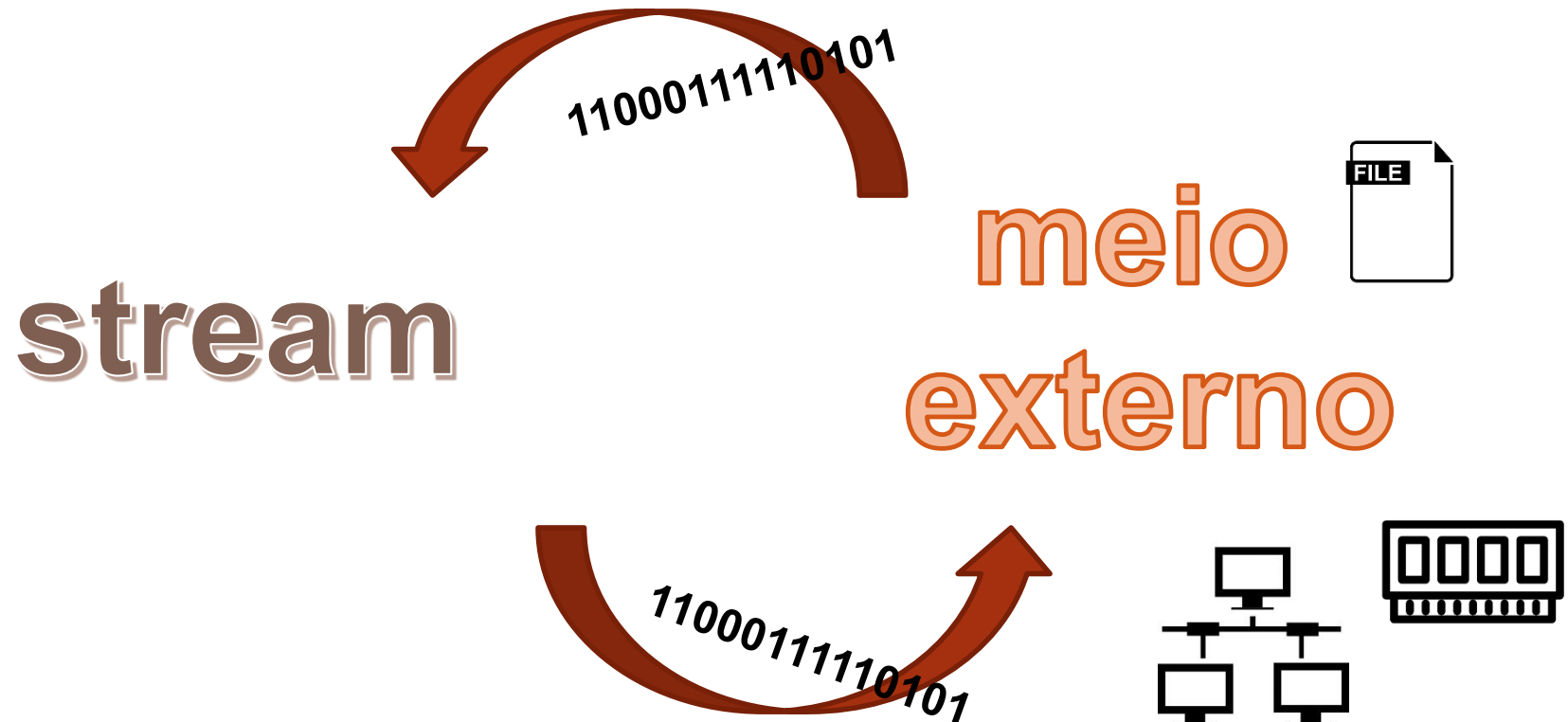
- Uma *stream* é como o Framework .NET transfere dados do seu programa para entidades externas (ou vice-versa), isto é, através de fluxo de dados.



Streams



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



Streams

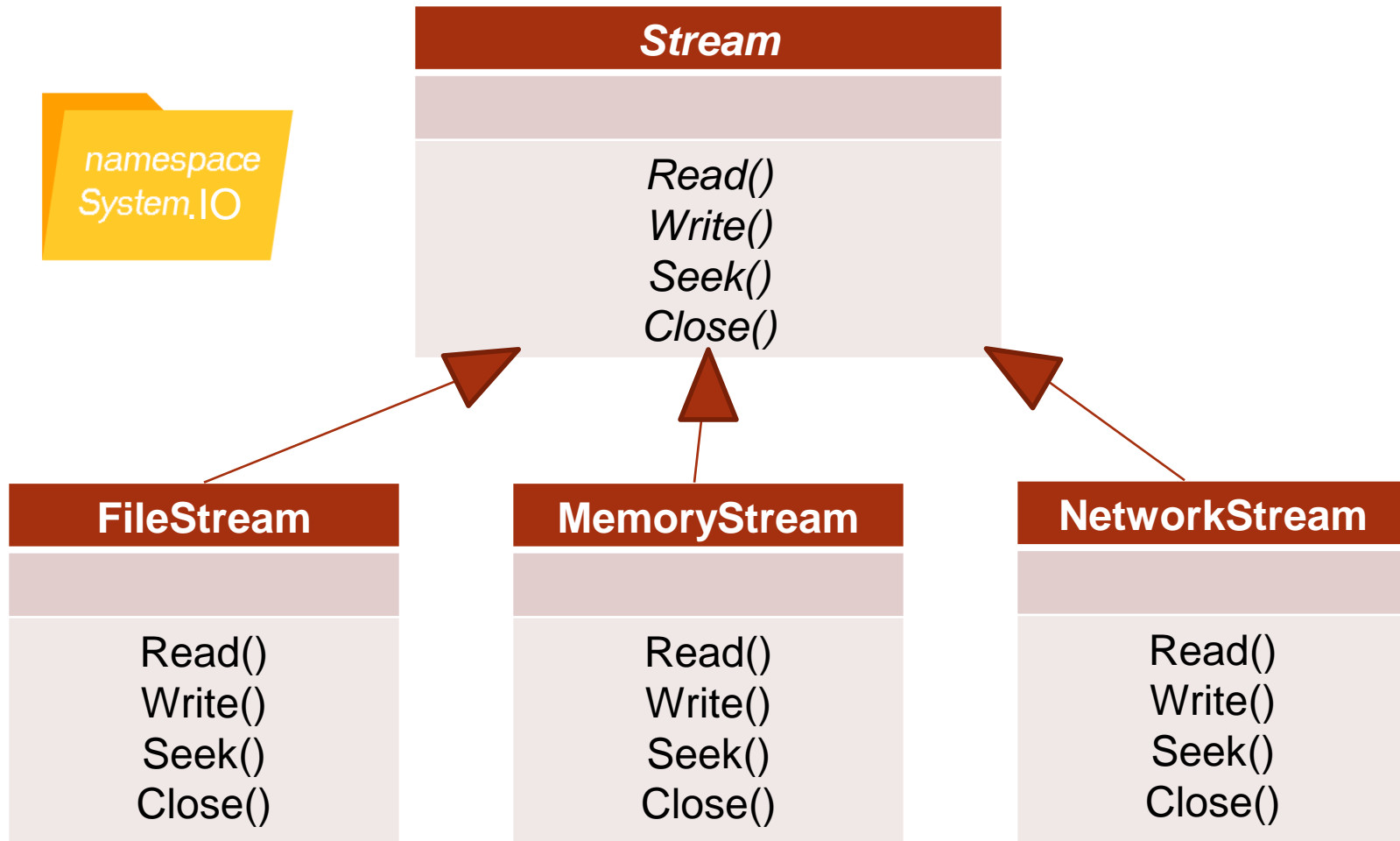
Classes .NET



- Há no .NET várias classes que realizam a tarefa de lidar com o exterior do seu programa. Algumas lidam com arquivos, outras lidam com outros computadores ou dispositivos de redes, etc, etc.
- De qualquer forma, todas essas classes que fazem comunicações diferentes com o meio exterior do seu programa são uma subclasse da classe abstrata *Stream*.
- Não somente, toda classe filha de *Stream* precisa implementar alguns métodos abstratos que são operações básicas de troca de dados: **escrever**, **ler**, **procurar** e **fechar**.
- Dentro da FCL, o namespace que contém essas classes é denominado **System.IO**.

Streams

Classes .NET



Streams

Operações básicas



1. **Escrever:** você pode escrever texto ou dados binários em uma stream através do método `Write()`.
2. **Ler:** você pode usar o método `Read()` para recuperar dados de um arquivo, de uma rede, da memória ou praticamente de qualquer outro lugar.
3. **Procurar:** a maioria das streams suporta um método `Seek()` (procurar) que permite encontrar uma posição dentro da stream, permitindo inserir ou ler dados em/de um local específico.
4. **Fechar:** liberar a conexão firmada entre a Stream e o arquivo externo.

FileStream



Iremos nos concentrar em ler e escrever em arquivos comuns, mas tudo aprendido nesta aula é aplicável a arquivos compactados ou encriptados ou streams de rede que não usem arquivos de forma alguma.



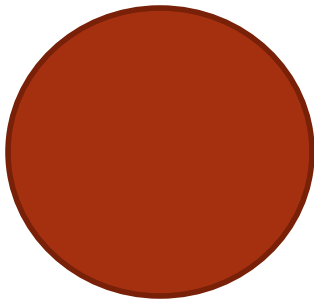
- Um **FileStream** escreve e recupera bytes de um arquivo.
- Quando seu programa precisa se comunicar com um arquivo externo, as seguintes etapas devem ser realizadas.

FileStream

Escrevendo em um arquivo



1. Crie um novo objeto FileStream.



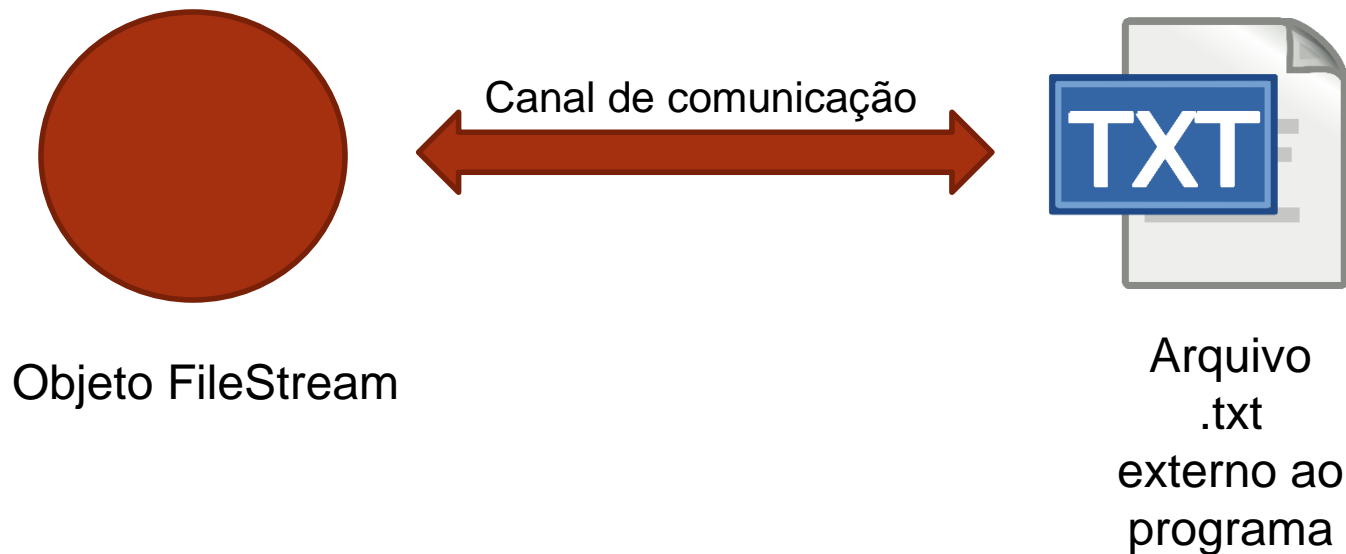
Objeto FileStream

FileStream

Escrevendo em um arquivo



2. Associe o objeto com um arquivo especificado em um caminho (*path*).

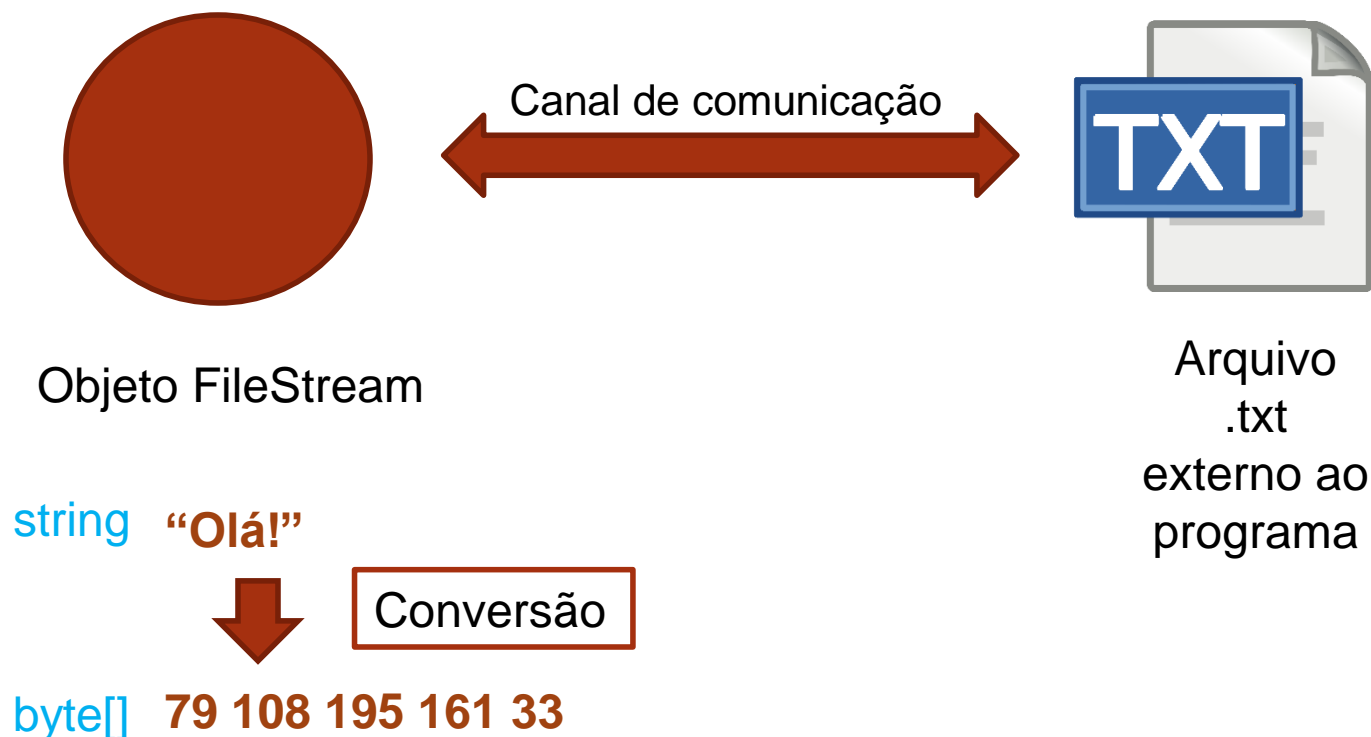


FileStream

Escrevendo em um arquivo



3. Streams escrevem bytes em arquivos. Logo, você precisará converter a sequência de caracteres que deseja escrever em um vetor de bytes.

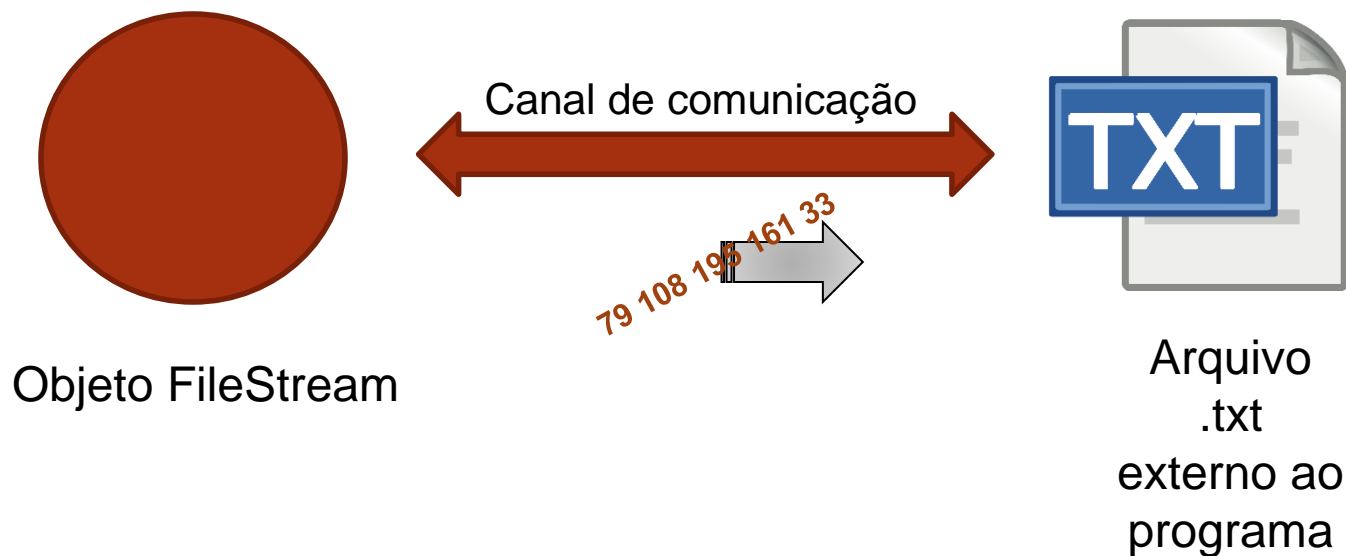


FileStream

Escrevendo em um arquivo



4. Chame o método Write() da stream e passe o vetor de bytes para ele.

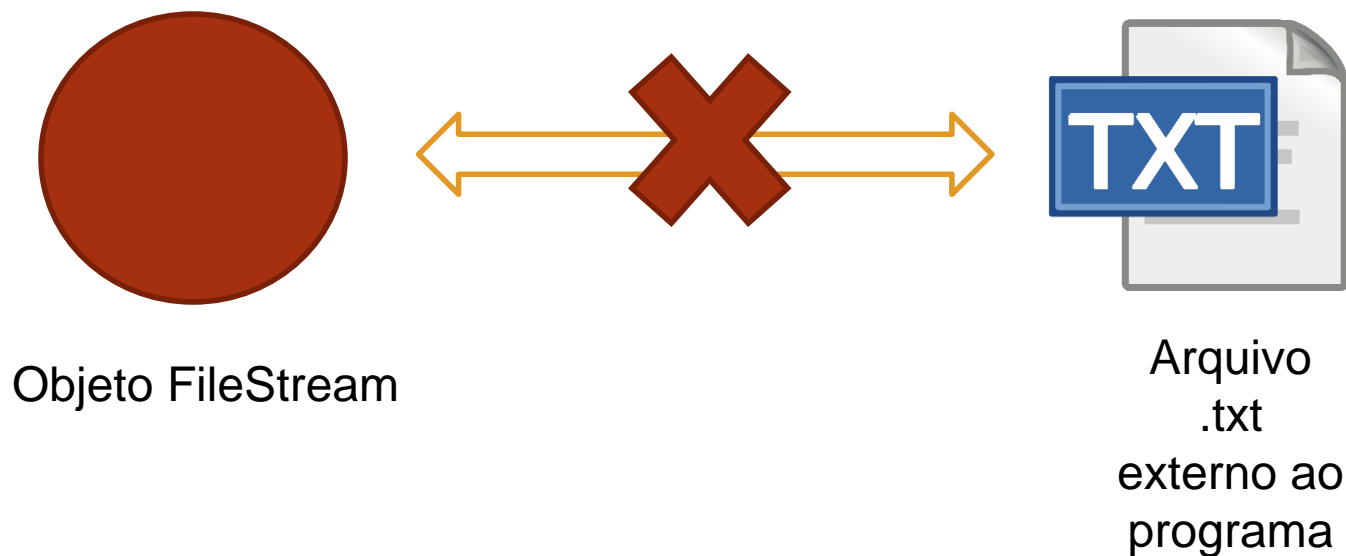


FileStream

Escrevendo em um arquivo



- 5.** Feche o arquivo para que outros programas possam acessá-lo. (Esquecer de fechar um stream é um grande problema. Se isso acontecer, o arquivo travará e outros programas pois eles não poderão usá-lo até você fechar sua stream.)



FileStream

Escrevendo em um arquivo



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

        // Deleta o arquivo se ele já existe no caminho especificado
        if (File.Exists(path))
        {
            File.Delete(path);
        }

        // Início da operação de escrita
        FileStream fs = File.Create(path);

        string text = "Olá mundo no arquivo!";

        byte[] info = new UTF8Encoding(true).GetBytes(text);
        fs.Write(info, 0, info.Length);

        fs.Close();
        // Fim da operação de escrita
    }
}
```

FileStream

Lendo de um arquivo



```
// Início da operação de leitura
fs = File.OpenRead(path);

byte[] b = new byte[1024];
UTF8Encoding temp = new UTF8Encoding(true);

while (fs.Read(b, 0, b.Length) > 0)
{
    Console.WriteLine(temp.GetString(b));
}

fs.Close();
// Fim da operação de leitura

Console.ReadLine();
}
```

FileStream

Boa notícia...



- Entendemos como a comunicação com um arquivo externo se dá, no entanto, o código não parece nada amigável.
- Ficar realizando conversão de bytes não parece que deveria ser uma tarefa do programador (e sim da plataforma de programação).
- Felizmente o C# provê uma outra classe para se comunicar com arquivos de um modo muito mais simplificado!

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



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



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



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



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



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



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.

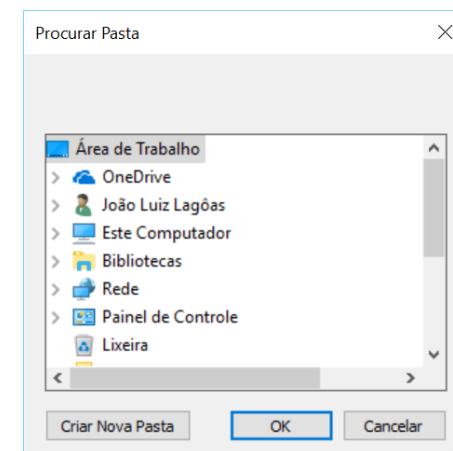
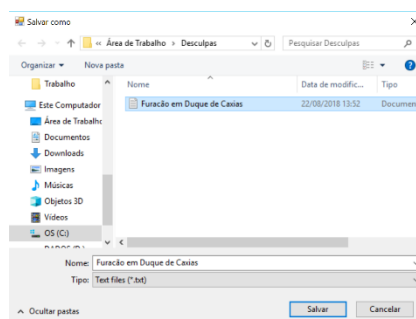
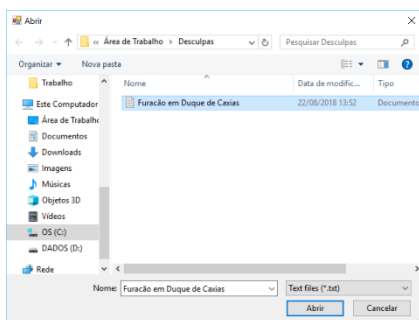
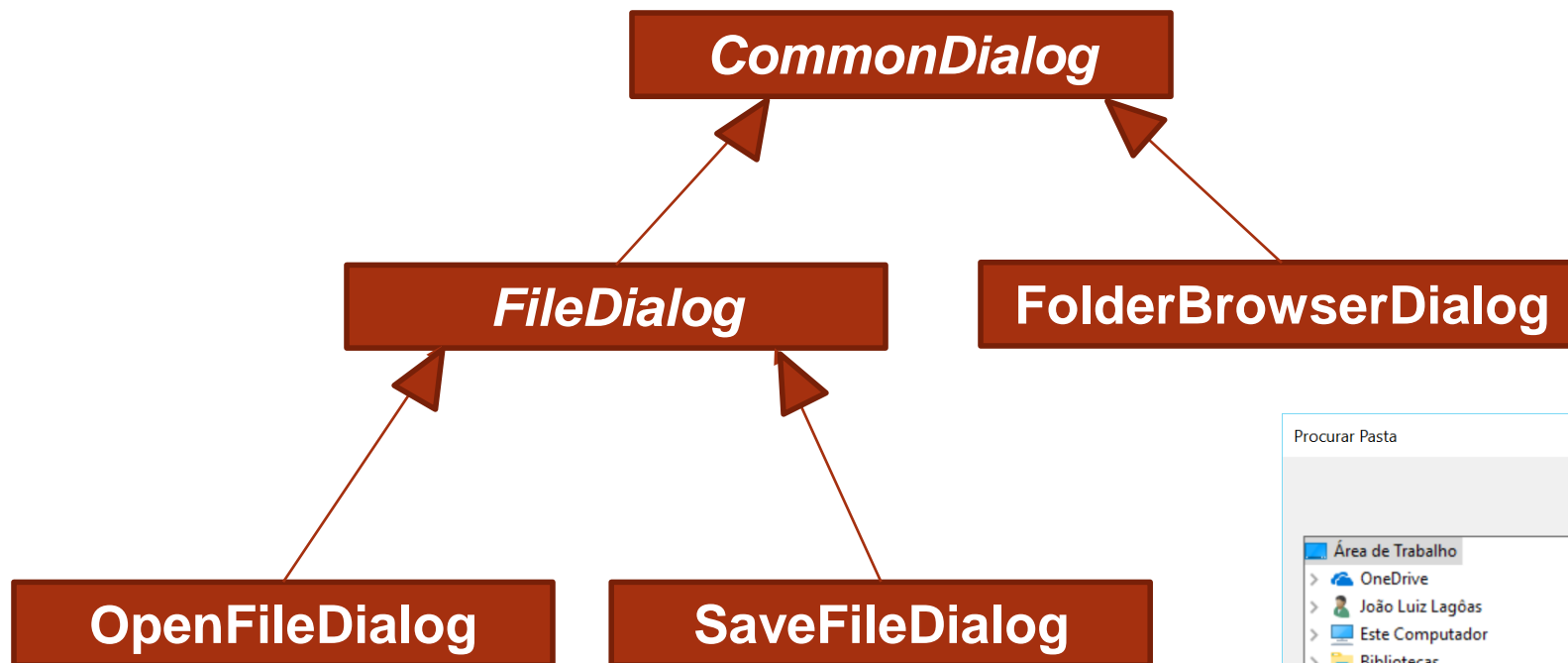
Caixas de Diálogo



- Ao se trabalhar com um programa que lê e escreve arquivos, existe uma boa chance de se precisar mostrar uma **caixa de diálogo** em algum momento para solicitar ao usuário um nome de arquivo e/ou um diretório.
- A plataforma .NET conta com vários tipos de caixas de diálogo para serem usadas em nossos programas. Nós estudaremos três delas:
 - FolderBrowserDialog
 - OpenFileDialog
 - SaveDialog

Caixas de Diálogo

Tipos de Dialog Boxes

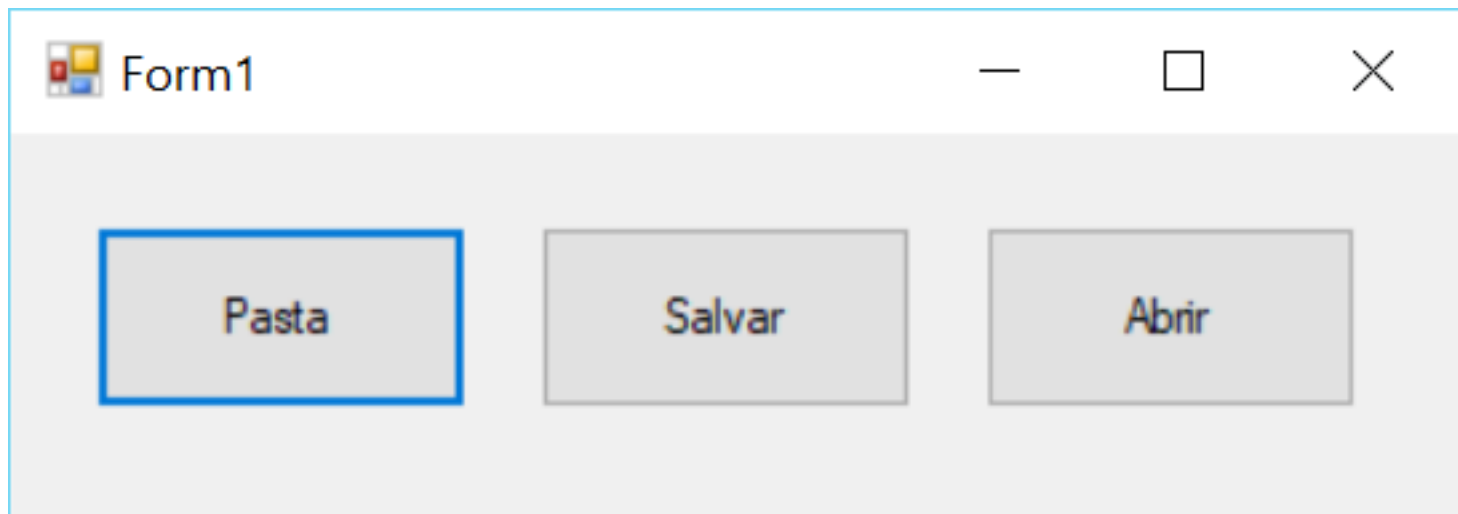


Caixa de Diálogo

Exemplo



- Vamos tomar o exemplo abaixo para entender como se é possível utilizar essas caixas de diálogo.



- Ao se clicar em cada botão, uma caixa de diálogo deve ser aberta com a respectiva função.

Caixa de Diálogo

OpenFileDialog



```
private void openButton_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();

    openFileDialog.InitialDirectory = @"C:\";
    openFileDialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";

    DialogResult result = openFileDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        MessageBox.Show("O usuário clicou em OK!");
        MessageBox.Show("O caminho do arquivo clicado é: " +
            openFileDialog.FileName);
    }

    else if (result == DialogResult.Cancel)
    {
        MessageBox.Show("O usuário clicou em Cancel!");
    }
}
```

Caixa de Diálogo

OpenFileDialog



- Um objeto OpenFileDialog mostra a janela de “Abrir” padrão do Windows.
- Para se usar um objeto OpenFileDialog, é comum seguir três passos:
 1. Criar uma instância da classe OpenFileDialog com o comando new.
 2. Alterar seus atributos de modo a personalizar seu objeto.
 3. Chamar o método ShowDialog() para exibir a janela na tela.

Nota: Esse método retorna uma variável do tipo DialogResult armazenando a opção de clique do usuário.

Nota: O atributo **FileName** retorna o caminho do arquivo carregado.

Caixa de Diálogo

SaveFileDialog



```
private void saveButton_Click(object sender, EventArgs e)
{
    SaveFileDialog saveDialog = new SaveFileDialog();

    saveDialog.InitialDirectory = @"C:\";
    saveDialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";

    DialogResult result = saveDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        MessageBox.Show("O usuário clicou em OK!");
        MessageBox.Show("O caminho do arquivo a ser salvo é: " +
            saveDialog.FileName);
    }

    else if (result == DialogResult.Cancel)
    {
        MessageBox.Show("O usuário clicou em Cancel!");
    }
}
```

Caixa de Diálogo

SaveFileDialog



- Um objeto `SaveFileDialog` mostra a janela de “Salvar” padrão do Windows.
- Para se usar um objeto `SaveFileDialog`, é comum seguir três passos:
 1. Criar uma instância da classe `SaveFileDialog` com o comando `new`.
 2. Alterar seus atributos de modo a personalizar seu objeto.
 3. Chamar o método `ShowDialog()` para exibir a janela na tela.

Nota: Esse método retorna uma enum `DialogResult` armazenando a opção de clique do usuário.

Nota: O atributo **FileName** retorna o caminho especificado do arquivo a ser salvo.

Caixa de Diálogo

FolderBrowserDialog



```
private void folderButton_Click(object sender, EventArgs e)
{
    FolderBrowserDialog folderDialog = new FolderBrowserDialog();

    //Pode mudar algum atributo de folderDialog aqui...

    DialogResult result = folderDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        MessageBox.Show("O usuário clicou em OK!");
        MessageBox.Show("O caminho da pasta clicada é: " +
                        folderDialog.SelectedPath);
    }

    else if (result == DialogResult.Cancel)
    {
        MessageBox.Show("O usuário clicou em Cancel!");
    }
}
```

Caixa de Diálogo

FolderBrowserDialog



- Um objeto FolderBrowserDialog mostra a janela de “Abrir Pasta” padrão do Windows.
- Para se usar um objeto FolderBrowserDialog, é comum seguir três passos:
 1. Criar uma instância da classe FolderBrowserDialog com o comando new.
 2. Alterar seus atributos de modo a personalizar seu objeto.
 3. Chamar o método FolderBrowserDialog() para exibir a janela na tela.

Nota: Esse método retorna uma enum DialogResult armazenando a opção de clique do usuário.

Nota: Observe também o atributo **SelectedPath** do objeto FolderBrowserDialog para verificar qual caminho foi selecionado.