



Gestion des Fichiers et des répertoires C#

Mme Ben bouzid NOURHENE



Plan:

- Introduction
- File, FileInfo and FileStream classes
- StreamReader et StreamWriter
- Directory Class
- Exercice

Introduction

- Le Framework .Net fournit l'espace de noms System.IO, qui contient un certain nombre de classes qui aident à simplifier les applications nécessitant des fonctionnalités d'E/S.
- le namespace System.IO contient des classes pour la manipulations des fichiers et des dossiers.

C#

```
using System.IO;
```

File Class:

La classe **File** Fournit des méthodes statiques pour créer, copier, supprimer, déplacer et ouvrir un fichier.

- Cette classe est utilisé pour les opérations courantes telles que la copie, le déplacement, le changement de nom, la création, l'ouverture, la suppression et l'ajout d'un seul fichier à la fois.
- Vous pouvez également utiliser la classe File pour obtenir et définir des attributs de fichier ou des informations relatives à la création, l'accès et l'écriture d'un fichier.
- Toutes les méthodes de cette classe requièrent le chemin d'accès au fichier que vous manipulez.

C#

```
public static class File
```

Héritage [Object](#) → File

Méthodes de la classe File:

<code>AppendAllLines(String, IEnumerable<String>)</code>	Ajoute des lignes à un fichier, puis ferme le fichier. Si le fichier spécifié n'existe pas, cette méthode crée un fichier, écrit les lignes spécifiées dans le fichier, puis ferme le fichier.
<code>AppendAllLines(String, IEnumerable<String>, Encoding)</code>	Ajoute des lignes à un fichier en utilisant un encodage spécifié, puis ferme le fichier. Si le fichier spécifié n'existe pas, cette méthode crée un fichier, écrit les lignes spécifiées dans le fichier, puis ferme le fichier.
<code>AppendAllText(String, String)</code>	Ouvre un fichier, ajoute la chaîne spécifiée au fichier, puis ferme le fichier. Si le fichier n'existe pas, cette méthode crée un fichier, écrit la chaîne spécifiée dans le fichier, puis ferme le fichier.
<code>AppendAllText(String, String, Encoding)</code>	Ajoute la chaîne spécifiée au fichier en utilisant l'encodage spécifié, en créant le fichier s'il n'existe pas.
<code>Copy(String, String)</code>	Copie un fichier existant vers un nouveau fichier. Le remplacement d'un fichier du même nom n'est pas autorisé.
<code>Copy(String, String, Boolean)</code>	Copie un fichier existant vers un nouveau fichier. Le remplacement d'un fichier du même nom est autorisé.
<code>Create(String)</code>	Crée ou remplace un fichier dans le chemin d'accès spécifié.
<code>Create(String, Int32)</code>	Crée ou remplace un fichier dans le chemin spécifié, en indiquant une taille de mémoire tampon.

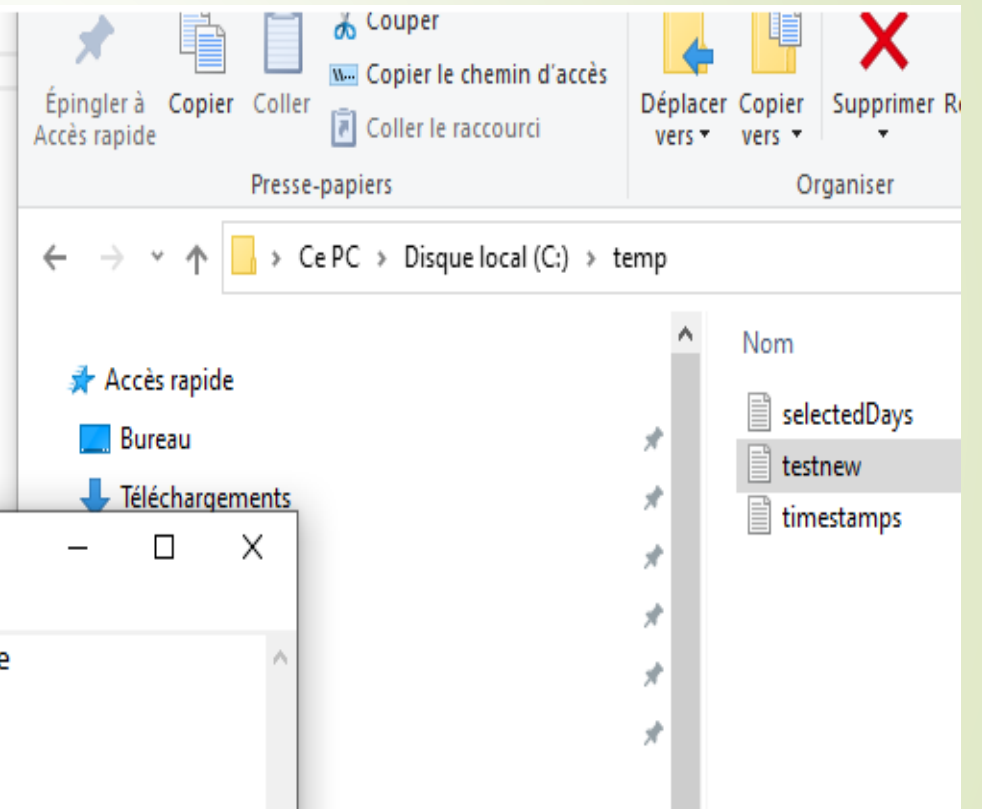
<code>Decrypt(String)</code>	Déchiffre un fichier qui a été chiffré par le compte actuel à l'aide de la méthode <code>Encrypt(String)</code> .
<code>Delete(String)</code>	Supprime le fichier spécifié.
<code>Encrypt(String)</code>	Chiffre un fichier de sorte que seul le compte utilisé pour chiffrer le fichier peut le déchiffrer.
<code>Exists(String)</code>	Détermine si le fichier spécifié existe.
<code>GetAttributes(String)</code>	Obtient l'élément <code>FileAttributes</code> du fichier sur le chemin d'accès.
<code>GetCreationTime(String)</code>	Retourne la date/heure de création du fichier ou du répertoire spécifié.
<code>GetLastAccessTime(String)</code>	Retourne la date/heure du dernier accès au fichier ou au répertoire spécifié.
<code>GetLastWriteTime(String)</code>	Retourne la date/heure du dernier accès en écriture au fichier ou au répertoire spécifié.
<code>GetLastWriteTimeUtc(String)</code>	Retourne la date/heure au format UTC (Temps universel coordonné) de la dernière écriture dans le fichier ou le répertoire spécifié.

Méthodes de la classe File:

<code>Move(String, String)</code>	Déplace un fichier spécifié à un nouvel emplacement, en permettant de spécifier un nouveau nom.
<code>Move(String, String, Boolean)</code>	Déplace un fichier spécifié vers un nouvel emplacement, en fournissant les options permettant de spécifier un nouveau nom de fichier et de remplacer le fichier de destination s'il existe déjà.
<code>Open(String, FileMode)</code>	Ouvre un <code>FileStream</code> sur le chemin spécifié avec un accès en lecture/écriture sans partage.
<code>ReadAllLines(String)</code>	Ouvre un fichier texte, lit toutes les lignes du fichier, puis ferme le fichier.
<code>ReadAllText(String)</code>	Ouvre un fichier texte, lit tout le texte du fichier, puis ferme le fichier.
<code>ReadLines(String)</code>	Lit les lignes d'un fichier.
<code>Replace(String, String, String)</code>	Remplace le contenu d'un fichier spécifié par le contenu d'un autre fichier, en supprimant le fichier d'origine et en créant une sauvegarde du fichier remplacé.
<code>Replace(String, String, String, Boolean)</code>	Remplace le contenu d'un fichier spécifié par le contenu d'un autre fichier, en supprimant le fichier d'origine, en créant une sauvegarde du fichier remplacé et en ignorant éventuellement les erreurs de fusion.
<code>WriteAllBytes(String, Byte[])</code>	Crée un nouveau fichier, écrit le tableau d'octets spécifié dans le fichier, puis ferme le fichier. Si le fichier cible existe déjà, il est remplacé.
<code>WriteAllLines(String, String[])</code>	Crée un nouveau fichier, écrit le tableau de chaînes spécifié dans le fichier, puis ferme le fichier.
<code>WriteAllText(String, String)</code>	Crée un nouveau fichier, écrit la chaîne spécifiée dans le fichier, puis ferme le fichier. Si le fichier cible existe déjà, il est remplacé.
<code>WriteAllText(String, String, Encoding)</code>	Crée un nouveau fichier, écrit la chaîne spécifiée dans le fichier en utilisant l'encodage spécifié, puis ferme le fichier. Si le fichier cible existe déjà, il est remplacé.
<code>WriteAllTextAsync(String, String, CancellationToken)</code>	Crée un fichier de façon asynchrone, écrit la chaîne spécifiée dans le fichier, puis ferme le fichier. Si le fichier cible existe déjà, il est remplacé.

File Classe Exemple :

```
using System.IO;
namespace File_testing
{
    Oréférences
    class Program
    {
        Oréférences
        static void Main(string[] args)
        {
            //File.Create(@"c:\temp\test.txt");
            File.AppendAllText(@"c:\temp\test.txt", "Here is the text to be added to the file");
            File.Move(@"c:\temp\test.txt", @"c:\temp\testnew.txt");
            Console.Read();
        }
    }
}
```



FileInfo Class:

Les méthodes statiques de la File classe effectuent des contrôles de sécurité sur toutes les méthodes.

Si vous envisagez de réutiliser un objet plusieurs fois, envisagez d'utiliser la méthode d'instance correspondante de la classe **FileInfo** , car la vérification de la sécurité n'est pas toujours nécessaire.

Cette classe permet les opérations courantes telles que la copie, le déplacement, le changement de nom, la création, l'ouverture, la suppression et l'ajout à des fichiers.

```
using System;
using System.IO;

class Program
{
    // Références
    static void Main(string[] args)
    {
        FileInfo fs = new FileInfo(@"c:\temp\test.txt");
        fs.Create();
        Console.WriteLine("File Created");
        Console.Read();
    }
}
```


FileStream Classe:

- Fournit un élément Stream pour un fichier, prenant en charge les opérations en lecture et en écriture aussi bien synchrones qu'asynchrones.
- On peut utiliser la classe FileStream pour lire, écrire, ouvrir et fermer des fichiers.
- Vous pouvez utiliser les Read ,Write, CopyTo...pour effectuer des opérations synchrones, ou ReadAsync , WriteAsync , CopyToAsync pour effectuer des opérations asynchrones.
- Pour pouvoir effectuer des opérations de fichier gourmandes en ressources sans bloquer le thread principal on utilise les méthodes asynchrones.

FileStream Class:

The image displays a Visual Studio IDE window with a C# program named `Program.cs` in the `fileStream` namespace. The code demonstrates the creation and use of the `FileStream` class. The `Main` method creates a new `FileStream` object pointing to `D:\\csharpfile.txt` with `FileMode.Create`, closes the stream, writes a message to the console, and reads a key from the console.

```
1 using System;
2 using System.IO;
3 namespace fileStream
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9
10             FileStream fs = new FileStream("D:\\csharpfile.txt", FileMode.Create);
11             fs.Close();
12             Console.WriteLine("File has been created and the Path is D:\\csharpfile.txt");
13             Console.ReadKey();
14         }
15     }
16 }
17
18
```

Below the code editor, a console window shows the output: `File has been created and the Path is D:\\csharpfile.txt`.

In the foreground, a Windows File Explorer window is open, showing the `Disque local (D:)` drive. The file `csharpfile` is visible in the list, with a modification date of `04/11/2021 11:52` and a size of `0 Ko`.

StreamReader et StreamWriter :

- Les classes StreamReader et StreamWriter sont utilisées pour lire et écrire des données dans des fichiers texte.
- Ces classes héritent de la classe de base abstraite **Stream**, qui prend en charge la lecture et l'écriture des octets dans un flux de fichiers.

StreamReader

- Permet de lire des lignes d'informations à partir d'un fichier texte standard.
- La valeur par défaut est l'encodage UTF-8, sauf indication contraire
- Voici quelques-unes de ses propriétés et méthodes:
- **Constructeur public** StreamReader(string path)
- **Propriétés**: public bool EndOfStream
- **Méthodes**: public string ReadLine() , public string Read () et Public void close()

StreamReader:

```
class FileApp {  
    static void Main(string[] args) {  
        try {  
            //Créez une instance de StreamReader pour lire à partir d'un  
            fichier  
            using (StreamReader sr = new StreamReader("c:/myFile.txt")) {  
                string line;  
  
                // Lire les lignes du fichier jusqu'à la fin.  
                while ((line = sr.ReadLine()) != null) {  
                    Console.WriteLine(line);  
                }  
            }  
        } catch (Exception e) {  
            Console.WriteLine("Le fichier n'a pas pu être lu.");  
            Console.WriteLine(e.Message);  
        }  
    }  
}
```

StreamWriter

- Permet d'écrire des caractères dans un flux avec un codage spécifique.
- La valeur par défaut est l'encodage UTF-8, sauf indication contraire
- Voici quelques-unes de méthodes:
- **Constructeur** : `public StreamWriter(string path)`
- **Méthodes** : `Close(); Write();` et `WriteLine();`

StreamReader et StreamWriter :

```
class FileApp {  
    static void Main(string[] args) {  
        string[] str = new string[]  
        {  
            "Lorem Ipsum",  
            "Lorem Ipsum",  
            "Lorem Ipsum",  
            "Lorem Ipsum"  
        };  
  
        using (StreamWriter sw = new StreamWriter("myFile.txt")) {  
  
            foreach (string s in str) {  
                sw.WriteLine(s);  
            }  
        }  
  
        // Lire et afficher chaque ligne du fichier.  
        string line = "";  
        using (StreamReader sr = new StreamReader("myFile.txt")) {  
            while ((line = sr.ReadLine()) != null) {  
                Console.WriteLine(line);  
            }  
        }  
    }  
}
```

StreamReader et StreamWriter :

```
FileStream stream = null;
try
{
    // Create a FileStream with mode CreateNew
    stream = new FileStream(fileName, FileMode.OpenOrCreate);
    // Create a StreamWriter from FileStream
    using (StreamWriter writer = new StreamWriter(stream, Encoding.UTF8 ))
    {
        writer.WriteLine("C# Corner Authors");
        writer.WriteLine("=====");
        writer.WriteLine("Monica Rathbun");
        writer.WriteLine("Vidya Agarwal");
        writer.WriteLine("Mahesh Chand");
        writer.WriteLine("Vijay Anand");
        writer.WriteLine("Jignesh Trivedi");
    }
}
```

StreamReader et StreamWriter :

- Visitez ces liens pour trouver toutes les propriétés et les méthodes des classes StreamReader et StreamWriter :

<https://docs.microsoft.com/fr-fr/dotnet/api/system.io.streamreader?view=net-5.0>

<https://docs.microsoft.com/fr-fr/dotnet/api/system.io.streamwriter?view=net-5.0>

FileStream VS StreamReader et StreamWriter :

- Lorsque'il s'agit de travailler avec du texte, nous devons prendre en compte l'encodage du texte.
- StreamReader et StreamWriter sont construits autour de la lecture de texte, ils gèrent donc l'encodage pour nous.
- Si vous instanciez StreamReader ou StreamWriter à l'aide du constructeur qui accepte un nom de fichier, il utilisera en fait FileStream en interne.



Directory Classe :

- La classe Directory expose des méthodes statiques pour créer, se déplacer et énumérer des répertoires et sous-répertoires.
- Cette classe ne peut pas être héritée.
- Utilisé pour les opérations courantes telles que la copie, le déplacement, le changement de nom, la création et la suppression de répertoires.
- Pour créer un répertoire, utilisez l'une des **CreateDirectory** méthodes.
- Pour supprimer un répertoire, utilisez l'une des **Delete** méthodes.
- Pour obtenir ou définir le répertoire actif d'une application, utilisez la **GetCurrentDirectory** ou **SetCurrentDirectory** méthode.

Directory Classe :


Quelques méthodes:

<code>CreateDirectory(String)</code>	Crée tous les répertoires et sous-répertoires dans le chemin d'accès spécifié, sauf s'ils existent déjà.
<code>Delete(String)</code>	Supprime un répertoire vide dans un chemin d'accès spécifié.
<code>EnumerateDirectories(String)</code>	Retourne une collection énumérable de noms complets de répertoires dans un chemin spécifié.
<code>EnumerateDirectories(String, String)</code>	Retourne une collection énumérable des noms complets de répertoires qui correspondent à un modèle de recherche dans un chemin spécifié.
<code>EnumerateFiles(String)</code>	Retourne une collection énumérable de noms de fichiers complets dans un chemin spécifié.
<code>EnumerateFiles(String, String)</code>	Retourne une collection énumérable des noms de fichiers complets qui correspondent à un modèle de recherche dans un chemin spécifié.

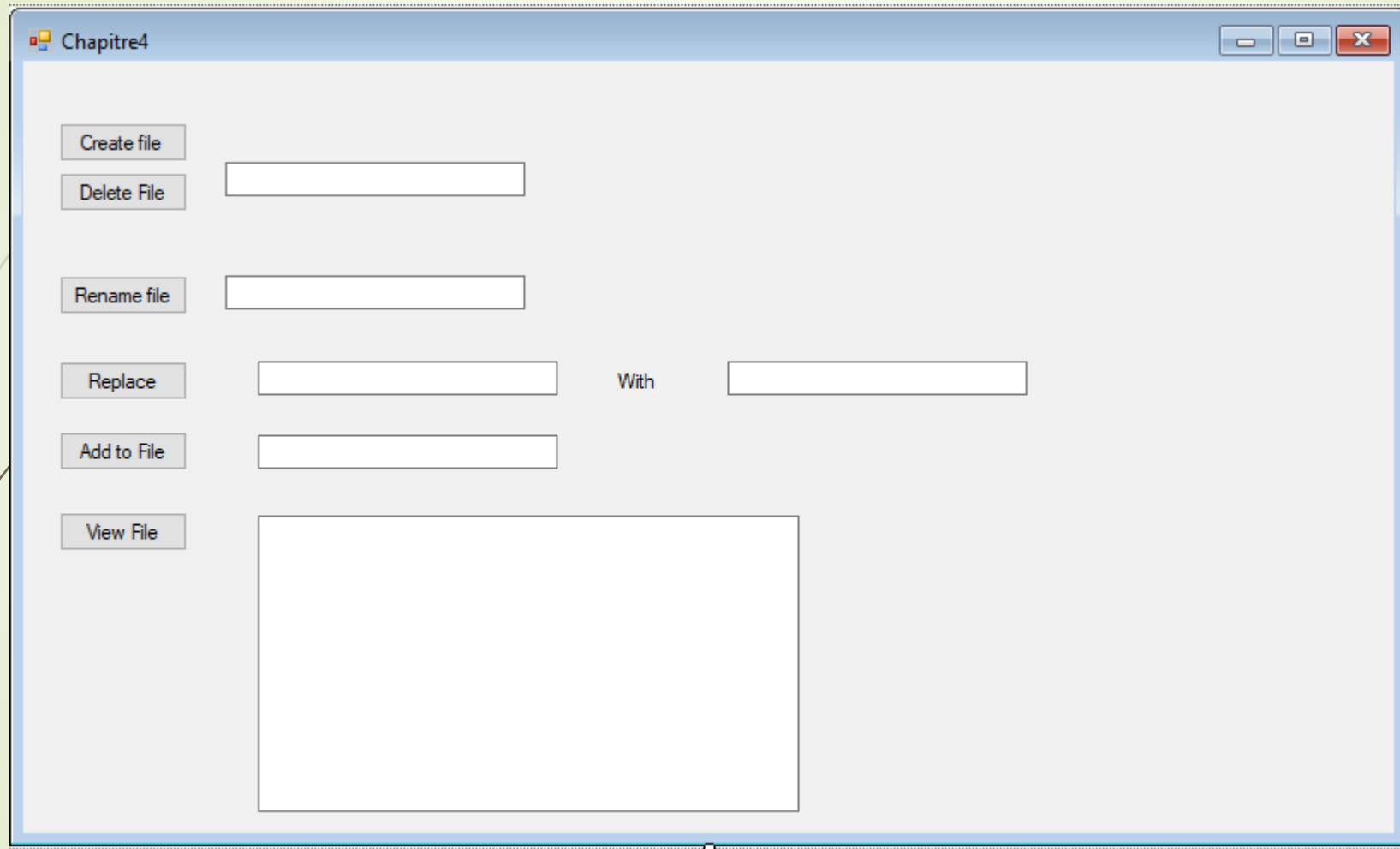
```
var txtFiles = Directory.EnumerateFiles(sourceDirectory, "*.txt");
```

<https://docs.microsoft.com/en-us/dotnet/api/system.io.directory?view=net-5.0>

Exercice :

- 
- Créer une application Windows Forms qui permet de:
 - Créer un fichier
 - Supprimer un fichier
 - Renommer un fichier
 - Remplacer une ligne spécifique dans un fichier
 - Afficher le fichier

Exercise :



The image shows a Java Swing window titled "Chapitre4" with standard window controls (minimize, maximize, close). The window contains a file management interface with the following elements:

- Create file**: A button.
- Delete File**: A button, followed by a single-line text input field.
- Rename file**: A button, followed by a single-line text input field.
- Replace**: A button, followed by a single-line text input field, the text "With", and another single-line text input field.
- Add to File**: A button, followed by a single-line text input field.
- View File**: A button, followed by a large multi-line text area.



Merci pour votre attention