

Structures de données et programmation orientée objet

COURS 13 – ÉCRITURE ET LECTURE DE FICHIERS

Introduction

Jusqu'à présent, nous avons toujours lu des intrants du programme à partir du clavier provenant directement de l'utilisateur, et avons écrit les résultats à l'écran. Ces deux dispositifs sont associés à la console, ce qui explique les instructions que nous avons utilisées :

- `Console.ReadLine()`
- `Console.WriteLine(...)`

Toutefois, lorsque la quantité de données à lire ou de résultats à écrire prend de l'ampleur, il est préférable de conserver les données dans des fichiers, afin de pouvoir les utiliser lors d'une prochaine exécution.

Introduction

Il est possible, à partir d'un programme, de lire dans un fichier plutôt qu'à partir du clavier. De la même manière, il est aussi possible d'écrire dans un fichier plutôt qu'à l'écran. Ainsi, les résultats d'une exécution peuvent servir de point de départ pour une prochaine exécution ou encore être conservés sur le disque tout simplement.

Ceci est une méthode plus idiomatique que d'agacer toujours l'utilisateur (qui souvent ne saurait pas quoi écrire, de toute façon).

Introduction

Dans le cadre de ce cours, nous nous intéresserons à un seul type de fichier : **les fichiers de texte**. Ces fichiers comportent du texte directement lisibles par l'utilisateur à l'écran au moyen d'un programme comme le Bloc-notes, ce qui simplifie la mise au point des programmes qui se servent de ce type de fichiers.

Lorsque les quantités de données deviennent vraiment importantes, nous passerons la plupart du temps par les services d'une base de données. Ce thème dépassant la portée de ce cours et nous ne l'aborderons pas.

Opération de base sur un fichier

Déclaration d'une variable représentant le nom logique du fichier à lire pour le programme, ainsi que le nom du fichier à écrire.

- `string nomFichierLecture = "Test.txt";`
- `string nomFichierEcrire = "Rapport.txt";`

Création de l'instance d'un objet permettant l'interaction avec le fichier

- Lecture : `StreamReader`
- Écriture : `StreamWriter`

Lecture dans un fichier d'entrée

Écriture dans un fichier d'entrée

Fermeture du fichier

La classe StreamReader

Pour qu'un programme puisse interagir avec un fichier sur le disque, nous pouvons créer un objet qui représente un fichier à lire.

Ces objets seront du type **StreamReader** (traduction : lecteur de flux)

Pour lire un fichier qui se trouve sur le disque, nous devons d'abord déclarer une référence sur une instance de la classe StreamReader.

- `StreamReader fluxLecture;`

TRÈS IMPORTANT : la classe StreamReader fait partie du namespace **System.IO**. Pour que ces symboles soient reconnus, le plus simple est d'ajouter en début de programme : **using System.IO;**

La classe StreamReader

Ayant déclaré une référence à un StreamReader, il nous faut l'instancier. Nous ferons appel au constructeur paramétrique de la classe par lequel nous spécifierons le nom du fichier à utiliser.

Par exemple, pour un fichier à lire, on pourrait avoir :

1. //Nom du fichier à lire
2. string nomFichierÀLire = "Test.txt";

3. //Déclaration d'une référence à un flux de lecture
4. StreamReader fluxLecture;

5. //Instanciation d'un flux de lecture
6. fluxLecture = new StreamReader(nomFichierÀLire);

La classe StreamReader

Par défaut, le chemin utilisé pour un fichier est le même que celui où se trouve l'exécutable.

Local Disk (F:) > Documents > Collège Lionel-Groulx > 202 > 2017 > Exemples > Exemple 9 - Lecture fichier texte > Exemple 9 - Lecture fichier texte > bin > Debug				
Name	Date modified	Type	Size	
Exemple 9 - Lecture fichier texte.exe.config	2017-03-13 4:38 PM	XML Configuratio...	1 KB	
Exemple 9 - Lecture fichier texte.vshost.exe	2017-03-13 4:38 PM	Application	23 KB	
Exemple 9 - Lecture fichier texte.vshost.e...	2017-03-13 4:38 PM	XML Configuratio...	1 KB	

Ce n'est pas très pratique lorsqu'on fait les tests d'exécution directement à partir de l'environnement de développement puisque les fichiers source sont généralement dans le même répertoire que nos fichiers .cs.

La classe StreamReader

Dans le cadre du cours, nous allons préférer conserver les fichiers dans le dossier où se trouvent les autres fichiers .cs du projet. Pour modifier le chemin d'accès au fichier que vous désirez lire ou écrire à partir de votre exécutable, nous pouvons fournir **un chemin relatif à évaluer à partir du dossier par défaut.**

Il faut se rappeler que « .. » correspond au **dossier parent** du dossier courant.

Donc, si le dossier avec notre exécutable (le dossier courant) est Debug, « ../ » serait bin et « ../../ » serait le dossier contenant notre code source (les fichiers .cs).

La classe StreamReader

Bref, pour lire un fichier texte qui est situé avec les fichiers .cs :

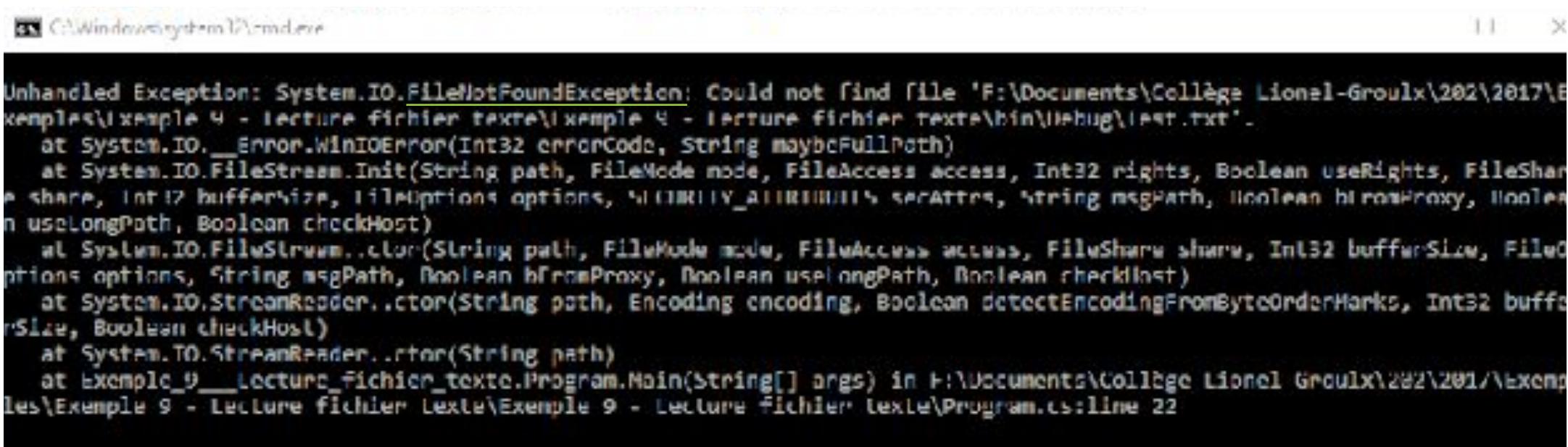
1. `//Chemin relatif où trouver les fichier`
2. `const string Chemin = "../../"`;
3. `//Nom du fichier à lire`
4. `string nomFichierÀLire = "Test.txt"`;

5. `//Déclaration d'une référence à un flux de lecture`
6. `StreamReader fluxLecture;`

7. `//Instanciation d'un flux de lecture`
8. `fluxLecture = new StreamReader(Chemin + nomFichierÀLire);`

La classe StreamReader

Si le fichier n'existe pas dans le dossier désigné, StreamReader lève une exception de système **FileNotFoundException** :



```
C:\Windows\system32\cmd.exe
Unhandled Exception: System.IO.FileNotFoundException: Could not find file 'F:\Documents\Collège Lionel-Groulx\2021\2017\Exemples\Exemple 9 - Lecture fichier texte\Exemple 9 - Lecture fichier texte\bin\Debug\test.txt'.
  at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
  at System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, Int32 rights, Boolean useRights, FileShare share, Int32 bufferSize, FileOptions options, String msgPath, Boolean bFromProxy, Boolean useLongPath, Boolean checkHost)
  at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share, Int32 bufferSize, FileOptions options, String msgPath, Boolean bFromProxy, Boolean useLongPath, Boolean checkHost)
  at System.IO.StreamReader..ctor(String path, Encoding encoding, Boolean detectEncodingFromByteOrderMarks, Int32 bufferSize, Boolean checkHost)
  at System.IO.StreamReader..ctor(String path)
  at Exemple_9__Lecture_fichier_texte.Program.Main(String[] args) in F:\Documents\Collège Lionel Groulx\2021\2017\Exemples\Exemple 9 - Lecture fichier texte\Exemple 9 - Lecture fichier texte\Program.cs:line 22
```

Lecture dans un fichier d'entrée

Nous savons déjà comment lire dans des flux d'entrée, à l'aide de la fonction ReadLine().

- `Console.ReadLine();` //Console est une classe statique représentant un flux d'entrée et de sortie, avec une méthode ReadLine pour lire une ligne de texte.
- `fluxLecture.ReadLine();` //fluxLecture est une instance de flux d'entrée avec une méthode ReadLine pour lire une ligne de texte.

ReadLine

Lorsqu'on lit un fichier à l'aide de ReadLine, le premier ReadLine va lire et retourner la première ligne dans le fichier. Le deuxième ReadLine lira la deuxième ligne. Le troisième lira la troisième ligne, et ainsi de suite...

Si un ReadLine nous retourne `null`, tout le fichier a été lu (ou le fichier était vide en partant...), car il n'y restait plus rien à lire.

Façon idiomatique de lire chaque ligne d'un fichier

En C#, nous suivons la syntaxe suivante pour ouvrir et fermer un flux de lecture :

```
1.  StreamReader fluxLecture;
2.  using (fluxLecture = new StreamReader(Chemin + nomFichierÀLire)
3.  {
4.      string ligne;
5.      while ((ligne = fluxLecture.ReadLine()) != null)
6.      {
7.          //Traitements avec l'information de la ligne...
8.          Console.WriteLine(ligne);
9.      }
10. } //Fermeture du flux
```

La raison d'utiliser `using` est que le flux va se fermer automatique et libérer sa prise sur le fichier, une fois sortie de la portée du *using*.

Séparer de l'information dans une chaîne de caractères

Souvent, nous voulons séparer plusieurs éléments dans une ligne de texte. Nous pouvons utiliser la méthode `Split(...)` sur une instance de `string` :

```
//texte avec 4 informations : prénom, nom, âge, salaire
string ligneDeTexte = "Bob,Sagat,42,32000";

//tableau des caractères qui délimitent chaque information
//ici, nous n'avons que la virgule
char[] délimiteurs = new char[1];
délimiteurs[0] = ',';

//Nous allons séparer toutes les informations de la ligne et les mettre dans un tableau
string[] informations;
informations = ligneDeTexte.Split(délimiteurs);
//La méthode Split retourne un tableau de string contenant les éléments de la chaîne à « splitter ».
//informations[0] <- "Yvon"  informations[1] <- "Rocher"
//informations[2] <- "42"    informations[3] <- "32000"
```

La classe StreamWriter

C'est bien de lire un fichier, mais c'est également important d'écrire dans un fichier !

Nous avons la classe **StreamWriter** (traduction : écrivain de flux), qui représente **un flux d'écriture**.

Pour écrire dans un nouveau fichier ou un fichier existant qui se trouve sur le disque, nous devons d'abord déclarer une référence sur une instance de la classe StreamWriter.

- **StreamWriter** fluxÉcriture;

TRÈS IMPORTANT : la classe StreamWriter fait partie du namespace **System.IO**.

La classe StreamWriter

L'utilisation d'un StreamWriter est semblable à celui d'un StreamReader :

```
1. StreamWriter fluxÉcriture;
2. using (fluxÉcriture = new StreamWriter(Chemin + nomFichierÀÉcrire)
3. {
4.     fluxÉcriture.WriteLine("blablabla");
5. } //Fermeture du flux
```

La classe StreamWriter

Si nous donnons un chemin vers un fichier *inexistant* au constructeur d'un StreamWriter, un fichier sera créé pour nous.

Si le fichier existe, le StreamWriter va écraser le contenu actuel du fichier avec un nouveau contenu. Si nous voulons écrire en partant de la fin du fichier, plutôt que de remplacer le contenu du fichier, nous pouvons donner un 2^e paramètre au constructeur :

-
1. `StreamWriter fluxÉcriture;`
 2. `using (fluxÉcriture = new StreamWriter(Chemin + nomFichierÀÉcrire, true)`
 3. `{`
 4. `fluxÉcriture.WriteLine("blablabla");`
 5. `} //Fermeture du flux`