C# : fichiers

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

C#

Les fichiers

- Moyen de stockage persistant permettant de conserver des données sur le disque dur.
- En **C#**, un fichier peut être :
  - lu (lecture),
  - créé ou modifié (écriture).

C#

La manipulation d'un fichier en **C#** suit généralement trois étapes :

- Ouverture ou création du fichier
- Utilisation (lecture ou écriture)
- Fermeture du fichier (libération des ressources)

© Achref EL

C#

La manipulation d'un fichier en **C#** suit généralement trois étapes :

- Ouverture ou création du fichier
- Utilisation (lecture ou écriture)
- Fermeture du fichier (libération des ressources)

© Achref EL

Remarque

En **.NET** moderne, la fermeture est généralement automatique grâce au mot-clé `using`.

C#

Première étape : création d'un objet `StreamWriter`

```
StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt");
```

© Achref EL MOUELHI ©

C#

Première étape : création d'un objet `StreamWriter`

```
StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt");
```

N'oublions pas d'utiliser l'espace de nom `using System.IO;`

C#

Remarques

- Le constructeur de la classe `StreamWriter` permet d'ouvrir le fichier dont le nom est passé en paramètre. Si le fichier n'existe pas, il sera créé.
- Si on n'a pas l'autorisation nécessaire pour la création du fichier, une exception sera levée.

C#

Entourons l'instruction précédente par un bloc `try ... catch ... finally`

```
try
{
   StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Utilisons la méthode `WriteLine` pour écrire dans le fichier puis la méthode `Dispose` pour le fermer

```
try
{
   StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt");
   sw.WriteLine("Hello World!!");
   sw.WriteLine("Bonjour tout le monde");
   sw.Dispose();
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Remarque

- `Dispose()` est la vraie méthode de libération des ressources.
- `Dispose()` est définie dans l'interface `IDisposable` et est conforme aux conventions **.NET** modernes.
- `Close()` est définie avant la généralisation de `IDisposable`.

C#

Nous pourrons aussi utiliser la méthode `Write` pour écrire sans faire de retour à la ligne (le résultat est le même)

```
try
{
   StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt");
   sw.Write("Hello ");
   sw.WriteLine("World!!");
   sw.Write("Bonjour tout le monde");
   sw.Dispose();
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Remarques

- À chaque exécution, le contenu précédent est supprimé et est remplacé par le nouveau.
- Le constructeur de la classe `StreamWriter` peut prendre un deuxième paramètre de type booléen qui permet d'indiquer si on veut
  - écrire à la suite dans le fichier si la valeur est `true`,
  - remplacer le contenu précédent si la valeur est `false` (par défaut).

C#

Ajoutons un booléen avec la valeur `true` comme deuxième paramètre au constructeur de la classe `StreamWriter` et lançons le programme plusieurs fois pour vérifier que le contenu s'ajoute plusieurs fois

```
try
{
   StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true);
   sw.Write("Hello ");
   sw.WriteLine("World!!");
   sw.Write("Bonjour tout le monde");
   sw.Dispose();
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Plusieurs surcharges pour les méthodes `Write` et `WriteLine`

```
try
{
   StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true);
   sw.WriteLine(5);
   sw.WriteLine(5.2);
   sw.WriteLine(true);
   sw.WriteLine('c');
   sw.Dispose();
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Le code précédent peut être simplifié en utilisant `using` [C# 8]

```
using (StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true)
{
   sw.WriteLine(5);
   sw.WriteLine(5.2);
   sw.WriteLine(true);
   sw.WriteLine('c');
   Console.WriteLine("Fin d'écriture dans le fichier");
}
```

© Achref El

C#

Le code précédent peut être simplifié en utilisant `using` [C# 8]

```
using (StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true)
{
   sw.WriteLine(5);
   sw.WriteLine(5.2);
   sw.WriteLine(true);
   sw.WriteLine('c');
   Console.WriteLine("Fin d'écriture dans le fichier");
}
```

© Achref El

Le fichier est automatiquement fermé à la fin du bloc `using` avec la méthode `Dispose`.

C#

Pourquoi utiliser `using` ?

- Ouverture ou création du fichier
- Utilisation (lecture ou écriture)
- Fermeture du fichier (libération des ressources)

C#

Avec la gestion des exceptions

```
try
{
   using StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true);
   sw.WriteLine(5);
   sw.WriteLine(5.2);
   sw.WriteLine(true);
   sw.WriteLine('c');
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Avec la gestion des exceptions

```
try
{
   using StreamWriter sw = new StreamWriter("C:\\Users\\elmou\\fichier.txt", true);
   sw.WriteLine(5);
   sw.WriteLine(5.2);
   sw.WriteLine(true);
   sw.WriteLine('c');
   Console.WriteLine("Fin d'écriture dans le fichier");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

Le fichier est automatiquement fermé à la fin du bloc `try` par `using`.

C#

Création d'un objet `StreamReader`

```
StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
```

© Achref EL MOUELHI

C#

Création d'un objet `StreamReader`

```
StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
```

MOUELHI

Remarques

- Le constructeur de la classe `StreamReader` permet d'ouvrir le fichier dont le nom est passé en paramètre.
- Si le fichier n'existe pas, une exception sera levée.

C#

Entourons l'instruction précédente par un bloc `try ... catch ... finally`

```
try
{
   StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Utilisons la méthode `ReadLine` pour lire la première ligne

```
try
{
   StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
   string line = sr.ReadLine();
   Console.WriteLine(line);
   sr.Dispose();
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Pour lire tout le contenu du fichier ligne par ligne

```
try
{
    StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
    string line = sr.ReadLine();
    while (line != null)
    {
        Console.WriteLine(line);
        line = sr.ReadLine();
    }
    sr.Dispose();
}
catch (Exception e)
{
    Console.WriteLine("Exception: " + e.Message);
}
finally
{
    Console.WriteLine("Bloc finally");
}
```

C#

Pour lire tout le contenu du fichier caractère par caractère, on utilise la méthode `Read` (qui retourne le code ASCII du caractère lu)

```
try
{
   StreamReader sr = new StreamReader("C:\\Users\\elmou\\fichier.txt");
   int c = sr.Read();
   while (c != -1)
   {
       Console.Write((char)c);
       c = sr.Read();
   }
   sr.Dispose();
}
catch (Exception e)
{
   Console.WriteLine("Exception: " + e.Message);
}
finally
{
   Console.WriteLine("Bloc finally");
}
```

C#

Remarque

`using()` peut aussi être utilisée pour la lecture de données d'un fichier.

© Achref EL HI ©

C#

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
15
12
17
13
10
16
```

© Achref

C#

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
15
12
17
13
10
16
```

© Achref

Exercice

Écrivez un code **C#** qui permet de calculer la moyenne des valeurs définies dans `notes.txt`.

C#

Autres opérations sur les fichiers

- `File.Create("chemin vers le fichier")` : pour créer un fichier
- `File.Delete("chemin vers le fichier")` : pour supprimer un fichier
- `File.Exists("chemin vers le fichier")` : retourne `true` si le nom du fichier passé en paramètre existe, `false` sinon.
- ...

C#

Opérations sur les répertoires

- `Directory.CreateDirectory("chemin vers le répertoire")` : pour créer un répertoire
- `Directory.Delete("chemin vers le répertoire")` : pour supprimer un répertoire
- `Directory.Exists("chemin vers le répertoire")` : retourne `true` si le nom du répertoire passé en paramètre existe, `false` sinon.
- `Directory.GetFiles("chemin vers le répertoire")` : retourne un tableau de chaînes de caractère correspondant aux noms de fichiers définis dans le répertoire passé en paramètre.
- ...