TP C#7

Rendu

Archive

Vous devez rendre un fichier zip avec l'architecture suivante :

```
– rendu-tp-prenom.nom.zip
        |– rendu-tp-prenom.nom/
                |– AUTHORS
                |– README
                |– Moulinette/
                        |– Moulinette.sln
                        |– Moulinette/
                                |– Program.cs
                                |– Moulinette.cs/
                                |– Rendu.cs
                                |– Exo.cs
                                |– Correction.cs
                                |– Tout sauf bin/ et obj/
```

— Il faut bien évidemment remplacer *prenom.nom* par votre nom.
— Le code doit compiler.
— Pas de dossiers bin ou obj.

AUTHORS

Ce fichier doit contenir : une étoile (*), un espace, votre login et une nouvelle ligne.
Voici un exemple (où $ represente une nouvelle ligne) :

```
* prenom.nom$
```

Merci de noter que les fichiers AUTHORS et README sont SANS extension.

Objectifs

Durant ce TP, nous allons aborder des nouvelles notions comme :
— L'exécution de commandes et programmes externes dans votre programme.
— Les flux standards (stdout et stderr).
— L'utilisation des List<T>.

1 Cours

1.1 Exécution de processus externe

L'un des principaux buts de ce sujet est d'exécuter des programmes/commandes dans votre programme C#. Pour cela, vous pouvez utiliser les classes **Process** et **ProcessStartInfo** (System.Diagnostics). Pour connaître toute les possibilités de ces classes (limiter l'espace mémoire, mettre un timeout, etc...), nous vous conseillons d'aller voir la documentation officielle (msdn). L'exemple ci-dessous exécute le programme "test.exe" situé dans le dossier "folder/".

```
1  private void Execute()
2  {
3      // Instanciate a new object of type ProcessStartInfo
4      ProcessStartInfo pStart = new ProcessStartInfo();
5      // Add the path of the executable to this object
6      pStart.FileName = "folder/test.exe";
7      // Disable window pop-up for the process
8      pStart.CreateNoWindow = false;
9      // Disable the shell of the OS to start the process
10     pStart.UseShellExecute = false;
11     // Run the executable
12     Process p = Process.Start(pStart);
13 }
```

1.2 Flux standard

En informatique, les flux standards sont des canaux de communication entre le programme et l'environnement où il est exécuté. Il en existe 3 :

— **Stdin**, pour l'entrée standard. Vous l'utilisez quand vous faites **Console.ReadLine()**.

— **Stdout**, pour la sortie standard. Vous l'utilisez avec **Console.WriteLine()**.

Afin de pouvoir récupérer les deux flux de sorties, nous avons modifié l'exemple précédent :

```
1  private void Execute()
2  {
3      ProcessStartInfo pStart = new ProcessStartInfo();
4      pStart.FileName = "folder/test.exe" ;
5      pStart.CreateNoWindow = false;
```

```
6       pStart.UseShellExecute = false;
7
8       //Enable the stdout getter of the process
9       pStart.RedirectStandardOutput = true;
10      //Enable the stderr getter of the process
11      pStart.RedirectStandardError = true;
12      Process p = Process.Start(pStart);
13      //Get the process stdout stream
14      string stdout = p.StandardOutput.ReadToEnd();
15      //Get the process stderr stream
16      string stderr = p.StandardError.ReadToEnd();
17  }
```

1.3 List<T>

List<T> est une collection standard de la bibliothèque .NET. C'est en réalité une implémentation optimisée des listes dynamiques. Le "T" correspond au type des éléments contenus dans la list : **List<int>** est une liste qui ne contient que des entiers, **List<string>** est une liste qui ne contient que des éléments de type string, etc.. Vous verrez le principe des templates plus tard dans l'année. Vous devez juste comprendre qu'on ne peut ajouter que des entiers dans une List<int>. Encore une fois, afin de connaître toutes les possibilités des listes en C# (obtenir le nombre d'éléments dans la liste, accéder à un élément, etc...), nous vous conseillons d'aller voir la documentation officielle (msdn).

Voici un exemple d'utilisation :

```
1   private void foo()
2   {
3       //Init a new list of int element
4       List<int> list = new List<int>();
5       //Add 1 on the list
6       list.Add(1);
7       //Add 2 on the list
8       list.Add(2);
9       for (int i = 0; i < list.Count; ++i)
10      {
11          //Get element at the i index
```

```
12          int n = list[i];
13          //Print on stdout the element
14          Console.WriteLine(n.ToString());
15      }
16  }
```

2 Exercice : Moulinette

2.1 Introduction

La Moulinette est un programme utilisé par vos ACDC (et sera utilisé par vos futurs ASM, ACU et YAKA), qui vérifie si votre code source fonctionne correctement. Elle compile, exécute et compare le résultat de votre programme avec ceux de la correction. Cette semaine, vous allez implémenter une mini Moulinette en C#. Votre Moulinette devra :

— Récupérer toutes les corrections.
— Récupérer tous les rendus et tous les exercices de chaque rendu.
— Exécuter les exercices (vous ne devez pas les compiler) et comparer les sorties (stdout et stderr) avec la correction.

Ne vous inquiétiez pas : ce TP n'est pas si compliqué que cela si vous avez correctement compris les classes et toute la partie cours (ce que c'est et comment l'utiliser). Vous pouvez (devez) utiliser les codes sources donnés dans la partie cours.

Vous pouvez considérer que le dossier où votre programme sera exécuté ressemblera à celui ci :

```
moulinette.exe
correction/
  |-- ex1/
    |-- stdout.txt
    |-- stderr.txt
  |-- ex2/
    |-- stdout.txt
    |-- stderr.txt
  |-- ...

rendu/
  |-- rendu-tp-login_x/
    |-- ex1/
      |-- exo.exe
    |-- ex2/
```

```
      |-- exo.exe
  |-- rendu-tp-login_y/
    |-- ex2/
      |-- exo.exe
  |-- ...
```

2.2 Classe : Correction.cs

Introduction

Cette classe va contenir le nom, le chemin (folder), et les flux de sorties (**stdout** et **stderr**) attendu d'un exercice.

Voici la liste des attributs dont vous aurez besoin pour la classe Correction :

```
1 // Attributs:
2 private string name;
3 private string folder;
4 private string stdout;
5 private string stderr;
```

Et maintenant, voici la liste des getter (retourne simplement la valeur des attributs) que vous devez avoir dans la classe **Correction** :

```
1 // Getters:
2 public string getName();
3 public string getStdout();
4 public string getStderr();
```

Le constructeur

Le constructeur de la classe **Correction** initialise UNIQUEMENT le nom et le chemin (folder) de l'objet (pas **stdout** et **stderr**). Le nom de l'exercice est le nom du dossier.

```
1 public Correction(string folderName);
```

La méthode Init

Cette méthode initialise les attributs **stdout** et **stderr** de l'objet. Pour ce TP, le flux **stdout** attendu est stocké dans le fichier stdout.txt et le flux **stderr** attendu est stocké dans le fichier stderr.txt. Elle retourne true si aucune erreur n'est détectée, false sinon.

```
1 public bool Init();
```

Indice : renseignez vous sur msdn à propos de **System.IO** pour allez lire dans un dossier (DirectoryInfo).

2.3 Classe : Exo.cs

Introduction

Cette classe contiendra tous les éléments associés à un exercice. Ici un exercice correspond au fichier exécutable fourni par l'étudiant. Il faut donc un attribut pour le chemin de l'exécutable.

Attributs et Getter

Voici la liste des attributs pour cette classe :

```
1 // Attributs
2 private string name;
3 private string folder;
4 private string stdout;
5 private string stderr;
```

A chaque attribut est associé un getter (retourne simplement la valeur de l'attribut privé).

```
1 // Getters
2 public string getName();
3 public string getFolder();
4 public string getStdout();
5 public string getStderr();
```

Constructeur

Le constructeur de cette classe initialise seulement les attributs "folder" et "name" de la classe Exo. En effet les attributs "stdout" et "stderr" seront initialisés seulement lors de l'exécution de l'exercice de l'étudiant.

```
1 public Exo(string folderName);
```

Execute

La méthode **Execute** permet sans surprise d'exécuter l'exercice de l'étudiant. Elle va ensuite récupérer les flux de sorties **stdout** et **stderr** afin de les stocker dans l'attribut associé. Cette méthode renvoie un booléen : **true** si aucun problème n'est détecté, et **false** en cas de problème (absence de l'exécutable de l'étudiant, ...).

```
1 public bool Execute();
```

Utiliser les exemples de la partie cours pour effectuer cet exercice !

2.4 Classe : Rendu.cs

Introduction

Dans ce TP un objet de type Rendu correspond au rendu d'un étudiant. Vous allez donc devoir instancier un objet de type Rendu pour chaque dossier d'étudiant. Tous les étudiants ont nommé leur exécutable "exo.exe".

Attributs

Les attributs de la classe Rendu sont les suivants :

```
1 private string folder;
2 private List<Exo> listExo;
```

L'attribut folder est une string contenant le chemin absolu du dossier de l'étudiant, et l'attribut listExo est une liste d'objet de type "Exo" que vous avez précédemment implémenté.

Constructeur

Le constructeur de la classe Rendu prend un seul argument : le nom du dossier de l'étudiant. Il faudra donc initialiser l'attribut "folder" avec ce qui a été passé en argument, et initialiser l'attribut listExo avec une liste vide.

```
1 public Rendu(string folderName);
```

Init

Cette méthode va parcourir tous les sous-dossiers du rendu de l'étudiant afin de trouver ses exercices. Chaque exercice trouvé sera ajouté à "listExo".

```
1 public bool Init();
```

La valeur de retour de cette méthode est **false** si aucun exercice n'à été trouvé, **true** sinon.

RunCorrection

Cette méthode doit pour chaque élément de la liste d'objet placé en paramètre :

- Chercher dans la liste des corrections l'exercice avec le même nom.
- Si aucun exercice n'est trouvé, afficher **nomDeLexercice + " : not found !\n"**.
- Si un exercice est trouvé, appeler la méthode **Execute()** de l'objet Exo.
- Si Execute() renvoie false afficher **nomDeLexercice + " : Exec error !\n"**.
- Si Execute() renvoie true comparer stdout et stderr avec la correction
- Si **stdout** et **stderr** sont correctes, afficher **nomDeLexercice + " : OK\n"** sinon afficher **nomDeLexercice + " : FAIL\n"**

```
1 public int RunCorrection(List<Correction> listCorrection);
```

La valeur de retour de cette méthode est le nombre de tests réussis.

2.5 Classe : Moulinette.cs

Introduction

Cette classe va contenir tous les rendu et toutes les corrections. C'est le cœur de notre moulinette, elle met en relation toutes les autres parties.

```
1 // Attributs:
2 private List<Correction> listCorrection;
3 private List<Rendu> listRendu;
```

Le constructeur

Le constructeur de la classe Moulinette initialise la liste de correction (listCorrection) et la liste de rendu (listRendu) en tant que liste vide. Elle peut aussi afficher quelques informations (comme "Moulinette v1.0").

```
1 public Moulinette();
```

La méthode init

Cette méthode remplit la liste de corrections (listCorrection) avec ce qui est contenu dans le dossier "correction" placé en racine, et la liste de rendu (listRendu) avec ce qui est contenu dans le dossier "rendu" placé lui aussi en racine. Vous ne devez ajouter dans les listes uniquement les objets dont leur méthode Init() renvoient true. Cette méthode renvoie true uniquement si les deux listes ne sont pas vides à la fin. Vous pouvez afficher le nombre d'éléments de chaque liste avant le return.

```
1 public bool Init();
```

La méthode Execute

La méthode Execute de la classe Moulinette appelle pour chaque objet de la liste de rendu (listRendu) la méthode **RunCorrection** et affiche le pourcentage de réussite.

```
1 public void Execute();
```

2.6 Main : Program.cs

Maintenant, toutes les classes dont vous avez besoin sont implémentées. Il ne vous reste plus qu'à écrire votre fonction **Main()** avec le code ci-dessous. Ce code DOIT fonctionner :

```
static void Main(string[] args)
{
  Moulinette moulinette = new Moulinette();
  if (moulinette.Init())
  {
    moulinette.Execute();
  }
}
```

Voici le résultat attendu (sur un terminal) :

```
--------------------
3 correction found!
3 rendu found!
--------------------
rendu-tp-login_x:
exo_0: OK
exo_1: KO
exo_2: Not found!
Grade: 33% (1/3)
--------------------
rendu-tp-login_y:
exo_0: OK
exo_1: OK
exo_2: Exec error!
Grade: 66% (2/3)
--------------------
rendu-tp-login_z:
exo_0: OK
exo_1: OK
exo_2: OK
Grade: 100% (3/3)
--------------------
```

The code is the law.