TP C#7: L'Empire contre attaque...

1 Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante :

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

Bien entendu, vous devez remplacer "`login_x`" par votre propre login. Dans le cas du rendu ci-dessus, l'élève s'appellerait par exemple Xavier Login ! :)

N'oubliez pas de vérifier les points suivants avant de rendre :
- Le fichier `AUTHORS` doit être au bon format (une *, un espace, votre login et un retour a la ligne). Sinon, c'est 0.
- Pas de dossiers `bin` ou `obj` dans le projet. Sinon, c'est 0.
- **Le code doit `ABSOLUMENT` compiler ! SINON, 0.**

2 Introduction

2.1 But du TP

Pendant la réalisation de ce TP, vous allez étudier les concepts suivants :
- l'exécution de commandes et programmes externes dans votre programme
- les flux standards (stdout et stderr)
- l'utilisation du conteneur List<T>

3 Cours

3.1 L'exécution de processus externe

L'un des principaux buts de ce sujet est d'exécuter des programmes/commandes dans votre programme C#. Pour faire cela, vous pouvez utiliser les classes Process et ProcessStartInfo (System.Diagnostics). Pour connaitre 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/".

```
private void execute()
{
        //Create a process configuration
        ProcessStartInfo pStart = new ProcessStartInfo();
        //Add the path of the executable
        pStart.FileName = "folder/test.exe";
        //Run the executable
        Process p = Process.Start(pStart);
}
```

3.2 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 : un en entrée (stdin, vous l'utilisez quand vous faites Console.ReadLine), deux en sorties (stdout (Console.WriteLine) et stderr (Console.Error.WriteLine)).

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

```
private void execute()
{
        ProcessStartInfo pStart = new ProcessStartInfo();
        pStart.FileName = "folder/test.exe" ;

        //Enable the stdout getter of the process
        pStart.RedirectStandardOutput = true;
        //Enable the stderr getter of the process
        pStart.RedirectStandardError = true;

        Process p = Process.Start(pStart);
        //Get the process stdout stream
        string stdout = p.StandardOutput.ReadToEnd();
        //Get the process stderr stream
        string stderr = p.StandardError.ReadToEnd();
}
```

3.3 List<T>

`List<T>` est un conteneur standard de la bibliothèque .NET. C'est en réalité une implémentation optimisée des listes dynamiques (comme les `vector` en C++). 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>`. Au même titre que pour la partie "L'exécution de processus externe" et afin de connaitre 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).

Nous vous donnons ci-dessous un exemple d'utilisation :

```
private void fun()
{
        //Init a new list of int element
        List<int> list = new List<int>();
        //Add 1 on the list
        list.Add(1);
        //Add 2 on the list
        list.Add(2);
        for (int i = 0; i < list.Count; ++i){
                //Get element at the i index
                int n = list[i];
                //Print on stdout the element
                Console.WriteLine(n.ToString());
        }
}
```

4 Moulinette

4.1 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 :

1. récupérer toutes les corrections
2. récupérer tous les rendus et tous les exercices de chaque rendu
3. 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/
|-- nameOfTheFirstExercice/
        |-- stdout.txt
        |-- stderr.txt
|-- nameOfTheSecondExercice/
        |-- stdout.txt
        |-- stderr.txt
|-- [...]
rendu/
|-- rendu-tp-login_x/
        |-- nameOfTheFirstExercice/
                |-- exo.exe
        |-- nameOfTheSecondExercice/
                |-- exo.exe
        |-- [...]
|-- rendu-tp-login_y/
        |-- nameOfTheFirstExercice/
                |-- exo.exe
        |-- nameOfTheSecondExercice/
                |-- exo.exe
        |-- [...]
|-- [...]
```

Avant de commencer à programmer, vous DEVEZ lire entièrement une première fois le sujet, et après cela, vous devez le relire une deuxième fois et seulement si vous avez compris tout le sujet, vous pouvez commencer, sinon relisez le tant que vous ne l'avez pas compris entièrement.

4.2 Correction.cs

4.2.1 Introduction

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

4.2.2 Attributs et Getter

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

```
private string name;
private string folder;
private string stdout;
private string stderr;
```

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

```
public string getName();
public string getStdout();
public string getStderr();
```

4.2.3 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.

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

4.2.4 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.

```
public bool init();
```

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

4.3 Exo.cs

4.3.1 Introduction

Cette classe va contenir le nom, le chemin (folder) et les contenus des flux stdout et stderr (après exécution) de l'exercice provenant d'un rendu d'un étudiant.

4.3.2 Attributs et Getter

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

```
private string name;
private string folder;
private string stdout;
private string stderr;
```

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

```
public string getName();
public string getStdout();
public string getStderr();
```

4.3.3 Le constructeur

Le constructeur de la classe Exo 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.

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

4.3.4 La méthode execute

Cette méthode exécute l'exercice. A la fin de l'exécution, elle va récupérer les flux de sortie (stdout et stderr) et les stocker dans l'objet. Pour ce TP, le nom du binaire à exécuter devra être "exo.exe". Si aucun binaire avec ce nom n'est trouvé, cette méthode doit retourner `false`. Si aucun problème n'est détecté, elle doit renvoyer `true`, `false` sinon.

```
public bool execute();
```

4.4 Rendu.cs

4.4.1 Introduction

Dans ce TP, un objet de type Rendu représente un rendu d'un étudiant. Chaque rendu contient 0 ou plus dossier. Chaque dossier représente un exercice. Chaque exercice contient un binaire avec le nom "exo.exe".

4.4.2 Attributs

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

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

4.4.3 Le constructeur

Le constructeur de la classe Rendu initialise le chemin du rendu (folder) et la liste des exercices du rendu (listExo) sans la remplir.

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

4.4.4 La méthode init

Cette méthode va initialiser la liste d'exercices (listExo) de l'objet. Souvenez vous : listExo est une liste d'objet Exo. Vous ne devez ajouter un objet dans la liste que lorsque sa méthode `init` renvoie `true`. Cette méthode renvoie `false` uniquement si aucun exercice n'a été trouvé.

```
public bool init();
```

4.4.5 La méthode runCorrection

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

- Chercher dans la liste d'exercice, l'exercice avec le même nom
- Si aucun exercice n'est trouvé, afficher `nomDeLExercice + ": not found!\n"`, sinon appeler la méthode `execute()` de l'objet Exo.
- Si execute() renvoit false, afficher `nomDeLExercice + ": error execute()!\n"`, sinon comparer stdout et stderr avec la correction
- Si stdout et stderr sont correcte, afficher `nomDeLExercice + ": OK\n"`, sinon afficher `nomDeLExercice + ": FAIL\n"`

Cette méthode retourne le nombre de tests réussis.

```
public int runCorrection(List<Correction> listCorrection);
```

4.5 Moulinette.cs

4.5.1 Attributs

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

```
private List<Correction> listCorrection;
private List<Rendu> listRendu;
```

4.5.2 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 "Moulnette v1.0").

```
public Moulinette();
```

4.5.3 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`.

```
public bool init();
```

4.5.4 La méthode execute

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

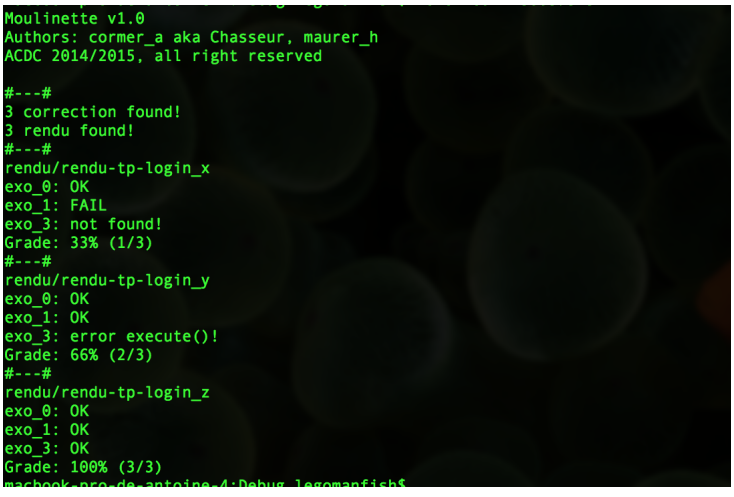
```
public void execute();
```

4.6 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();
        }
}
```

Quand votre ACDC exécutera votre code, il DOIT avoir quelque chose qui ressemble à cela :



5 Bonus

Avant de faire des bonus, vous devez avoir fait toute la partie obligatoire. Si cela est fait, alors vous pouvez vous lancer. Vous n'avez aucune limite, mais vous devez documenter tout ce que vous avez fait dans le fichier README. Si vous ne respectez pas cela, même avec toute la partie obligatoire, vous aurez 0.

The bullshit is strong with this one...