TP Plugin

On souhaite écrire une application modulaire de lecture audio. Cette application lit des fichiers audio encapsulés dans un fichier XML. La balise XML racine s'appelle *audio* et contient l'attribut *format* qui indique le format du fichier audio à lire.

Exemple :

```
<audio format='mp3'>
  <data>flux audio</data>
</audio>
```

La gestion des différents formats sera effectuée par des plugins chargés dynamiquement.

Créez un projet C# appelé « AudioPlayer ».
Ajoutez un fichier XML correspondant à un jeu d'essai, modélisez (pour l'instant) le flux audio par une chaine de caractères.

Ajoutez à l'application une classe Player et une classe IncorrectFileException dérivant d'Exception.

Cette exception pourra ainsi être levée à chaque anomalie détectée dans le fichier XML. Généralement, lorsqu'on définit ses propres exceptions, on définit au moins les mêmes constructeurs qu'`Exception` (pour être homogène) : `Exception()`, `Exception(String)` (permet de passer un message en paramètre) et `Exception(String, Exception)` (permet de passer un message et l'exception qui a générée l'exception, si besoin est).

Ajoutez à la classe Player la méthode Play qui prend un nom de fichier en argument. Cette méthode parse le fichier XML et affiche le nom du plugin à utiliser, en fonction du format spécifié dans le fichier.

La façon la plus simple de charger en mémoire un fichier XML est d'utiliser la classe `System.Xml.XmlDocument`. Ensuite, on joue avec la classe `XmlNode`. On récupère l'élément racine du document avec la propriété `XmlDocument.DocumentElement`. Le fichier XML se présente alors comme un arbre en mémoire avec des nœuds qui comportent chacun des attributs, et un contenu qui peut être soit une valeur sous forme de chaine de caractère soit des nœuds fils.

Créez une interface IAudioPlugin, qui sera implémentée par les plugins audio. Cette interface contient une méthode Decode qui prend (pour l'instant) une string en paramètre.

Créez une classe attribut .NET appelée « FormatAttribute » (voir TP introspection). Cet attribut sera utilisé par les classes plugin pour indiquer le format qu'elles sont capables de lire. Cet attribut comprend un membre de type string.

Ajoutez à votre solution un nouveau projet de type library et nommez-le MP3Plugin. La seule classe de ce projet implémente IAudioPlugin (pensez à ajouter une référence vers le projet AudioPlayer pour y avoir accès). La méthode « Decode » implémentée dans cette classe affiche : « mp3 » et la chaine de caractère passée en paramètre sur la console.

Précisez à l'aide de l'attribut .NET « FormatAttribute », au niveau de la classe, le format qu'est capable de lire ce plugin. Compilez la dll.

Dans le projet "AudioPlayer", ajoutez un répertoire pour stocker les plugins. Copiez-y la dll MP3Plugin compilée.

Implémentez une classe Factory. Cette factory contient deux méthodes. La première, la méthode « Register », prend en paramètre une chaine de caractère et une classe. Cette méthode permet d'enregistrer dans la factory une classe associée à une String. L'enregistrement se fait par une association clef - valeur dans une table de hashage. La deuxième méthode de cette classe est « GetInstanceByName ». Elle prend une string en paramètre et retourne un objet. La string permet de repérer le type d'objet à construire dans la table de hashage. La factory construit alors l'objet et le retourne.

La construction par défaut d'une instance à partir d'un objet `Type` peut se faire de la façon suivante :

```
monType.GetConstructor(Type.EmptyTypes).Invoke(new Object[0]);
```

Implémentez une exception IncorrectPluginException qui permet de signaler un plugin foireux, lors du chargement de celui-ci.
<optionnel>On souhaite être sur que la factory ne comprendra qu'une instance dans chaque instance de l'application, transformez-la par conséquent en singleton </optionnel>.
Ajoutez une méthode à la classe Player, nommée LoadPlugin, qui prend en paramètre un chemin vers un plugin sous forme de string. Cette méthode charge l'assembly et l'inspecte. Elle enregistre dans une factory la (ou les, si l'assembly contient plusieurs IAudioPlugin) classe(s) de décodage, en utilisant comme clé le format supporté par le plugin.

Modifiez la méthode Play pour que celle-ci instancie le bon type de plugin en fonction de l'attribut XML « format » et apelle la méthode de lecture du plugin sur les données.

On peut maintenant placer plein de plugins dans le repertoire *plugins*. Au démarrage de l'application, on parcourt le répertoire et on inscrit les plugins dans la factory. Il n'est donc plus necessaire de recompiler l'application pour l'étendre, un vrai progrès...

Question bonus pour ceux qui vont vite :

Pour l'instant, notre lecteur audio ne lit pas grand chose, il ne fait qu'afficher des chaînes de caractères à l'écran...C'est un peu la loose...Le framework .Net propose une classe `System.Media.SoundPlayer` qui permet de lire des données au format .wav. Implémentez un plugin `WAVPlugin` qui gère le .wav. Il faudra apporter quelques modifications à votre code :

- les données passées au plugin ne devront plus être sous forme de chaîne de caractères, mais sous forme de données binaires (tableau d'octets) ;
- le fichier XML contiendra les données binaires, sous forme textuelle (en base 64 par exemple) (J'ai mis en ligne un petit programme qui permet d'encapsuler un fichier binaire dans un fichier XML avec le format décrit dans ce TP : http://fc.isima.fr/~touraill/fic/upload/EmbedInXML.exe ; si vous voulez jeter un coup d'oeil au code, c'est dans le .tar à côté ; j'ai également uploadé un exemple de fichier : http://fc.isima.fr/~touraill/fic/upload/wavfile.xml)