TP C# 12

1 Sérialisation

1.1 Qu'est-ce que la sérialisation ?

> « *La sérialisation peut être définie comme étant le processus de stockage de l'état d'un objet sur un support de stockage. Pendant ce processus, les champs publics et privés de l'objet et le nom de la classe, y compris l'assembly contenant la classe, sont convertis en un flux d'octets, qui est alors écrit dans un flux de données. Lorsque l'objet est ensuite désérialisé, un clone exact de l'objet d'origine est créé.* »
>
> Msdn Microsoft

En d'autres termes, il s'agit de la possibilité de stocker un objet (une instance de classe) sur votre disque dur, sous forme binaire. Vous aurez ainsi la possibilité de le restaurer par la suite si vous voulez de nouveau l'utiliser.

Prenons un exemple concret d'utilisation. Imaginons que vous développiez un jeu vidéo en C# (*clin d'œil*) et que vous soyez en train de coder un éditeur de map. Vous avez donc créé une classe représentant votre map. Celle-ci contient divers attributs tels que les positions des éléments sur la map, etc..

Vous voulez maintenant sauvegarder votre map sous forme de fichier. Deux choix s'offrent a vous :
- Stocker le contenu de votre map actuelle sous une forme ou une autre, en faisant apparaitre tous les attributs dans ce fichier (sous forme de texte par exemple). Tout ceci à la main, vous obligeant à écrire un par un les attributs dans votre fichier.
- Utiliser la classe sérialisation binaire afin d'écrire directement votre objet dans un flux (vers un fichier par exemple). Tout ceci en 2-3 lignes de code. (sachant que la lecture du fichier pour récupérer votre objet sera tout aussi rapide)

Avouez que le deuxième choix a l'air intéressant non ? Bien évidemment l'utilisation de la sérialisation ne se limite pas à ce cas de figure :) Voyons maintenant comment ça marche.

1.2 You are a Serial killer !

Commençons par préparer le terrain :
- Créez une nouvelle solution console que vous nommerez : **my_serialisation**.
- Dans le fichier principal, créez une nouvelle classe nommée : **MyClass**.
- Créez un attribut de type string dans votre class.
- Créez un constructeur auquel on donnera une string en argument, et qui initialisera l'attribut avec celle-ci.
- Insérez les directives suivantes en haut du fichier principal :
  - using System.Runtime.Serialization ;
  - using System.IO ;
  - using System.Runtime.Serialization.Formatters.Binary ;

Il va maintenant falloir indiquer que votre classe peut être sérialisée en lui ajoutant l'attribut [Serializable], comme ceci :

```
[Serializable]
public class MyClass
...
```

Le code suivant permet de stocker une instance de la classe MyClass dans le fichier MyClass.bin :

```
// On crée une instance de notre classe MyClass
MyClass obj = new MyClass ("Foo");

// On crée un object de type BinaryFormatter qui permet de
// sérialiser un objet
IFormatter formatter = new BinaryFormatter ();

// On crée le fichier dans lequel l'objet sera stocké
Stream stream = new FileStream("MyFile.bin", FileMode.Create);

// On stocke notre objet dans le fichier
formatter.Serialize (stream, obj);
stream.Close ();
```

Exercice 1 - En utilisant ce code, créez une fonction générique de sérialisation, qui prendra n'importe quel objet en argument, ainsi qu'une string indiquant le nom du fichier à créer et qui sérialisera l'objet dans ce fichier. Voici le prototype :

```
public static void serializer(object o, string file);
```

Exercice 2 - Vous allez maintenant devoir désérialiser votre objet. Mais cette fois nous ne vous donnons pas le code (néanmoins, celui-ci est très semblable au précédent). Vous devrez :

- Créer un objet de type *BinaryFormatter*.
- Créer un FileStream vers le fichier.
- Utiliser la méthode *Deserialize* de l'objet BinaryFormatter.

Il vous faudra certainement utiliser un *cast* afin d'indiquer le type de l'objet renvoyé par le BinaryFormatter.

Voici le prototype de la fonction :

```
public static object deserializer(string file);
```

Exercice 3 (Bonus) - Dans les deux fonctions précédentes, vous devrez éventuellement catcher [1] les erreurs dans les cas suivants :

- Un objet ne possédant pas l'attribut [Serializable] est passé en argument à votre fonction *serializer*.
- Un fichier ne contenant pas un objet correct est passé à votre fonction *deserializer*.

En cas d'erreur, affichez celle-ci dans la console.

1. souvenez vous du bloc try... catch !

2 La réponse

2.1 One does not simply find the answer ..

Attention : Pour cette partie, utilisez la solution fournie "DeepThought.sln" accessible à l'adresse suivante : http ://perso.epita.fr/ guyotd_r

Contexte - Vous êtes un scientifique. Mais pas n'importe quel scientifique, vous êtes un des meilleurs scientifiques de votre race, qui est, entre parenthèses, la plus évoluée de l'univers. Il y a quelques années on vous a confié une mission simple : Construire un ordinateur d'une puissance incroyable, capable de calculer la réponse à la vie, à l'univers et au reste. Durant des années, vous avez construit cette incroyable machine avec votre équipe. Lorsqu'elle fut enfin fonctionnelle, elle fut mise en route, afin d'accomplir la mission pour laquelle elle avait été créée.

Mais la réponse a une volonté propre, et au bout de nombreuses années, il se produit une chose, une chose à laquelle elle ne s'attendait pas.. elle fut trouvée par l'être le plus insignifiant qui soit, un ordinateur. Mais l'esprit d'un ordinateur, aussi puissant soit-il, est aisément corruptible. La réponse le rongea, jusqu'à le convaincre de garder cette réponse pour lui...

C'est là que vous intervenez, vous, son créateur. Vous êtes le seul capable de retrouver la réponse, enfouie au fond de ce tas de ferraille ! Ce dernier n'a pas hésité à crypter, et recrypter la réponse afin qu'elle soit introuvable. Heureusement, vous avez quand même réussi à trouver quelques indications vous permettant de commencer vos recherches.

Vous avez réussi à trouver les fichiers suivants :
- Un fichier contenant un objet sérialisé.
- Un fichier crypté contenant la classe de l'objet sérialisé. Cet objet contiendra, entre autres, des attributs contenant des informations sur la réponse et sur les moyens de l'atteindre.

De plus, il vous est gracieusement fourni une solution contenant un fichier "Fixme.cs" contenant des fonctions que vous devrez compléter pour pouvoir décoder la réponse que vous cherchez.

2.2 Exercice : Décrypter le fichier DeepThought.cs

Le fichier contenant la classe a été crypté à l'aide d'un Rotn, ou chiffrement de césar (on a ajouté au code ASCII [2] de chaque caractère un certain nombre afin d'obtenir un autre caractère). Seulement vous ne connaissez pas le décalage utilisé. Il va donc vous falloir tester toutes les possibilités. Notez que pour ce chiffrement, une minuscule reste une minuscule, une majuscule reste une majuscule et un chiffre reste un chiffre même après une rotation de plusieurs dizaines de caractères. Les caractères spéciaux (espaces, retours à la ligne, tabulations, underscores...) ne sont pas modifiés. Vous devrez coder les deux fonctions suivantes :

Une fonction qui prendra en argument une chaine "message" ainsi qu'un entier "n" et qui devra renvoyer une chaine sur laquelle on aura appliqué le rotn.

```
public static char[] rotn(char[] message, int n);
```

A l'aide de la fonction précédente, vous allez pouvoir décrypter le fichier contenant la classe DeepThought. codez la fonction decrypt() dont le prototype est le suivant :

```
public static void decrypt(string filename);
```

Nous vous laissons libre dans le choix de l'implémentation de cette fonction. Elle doit vous permettre de trouver la bonne rotation a effectuer sur le fichier "DeepThought.cs" présent dans le dossier *bin/debug/* de la solution pour connaitre le contenu de la classe "DeepThought". Un fois ceci fait, ajoutez la classe décryptée à votre projet.

2. table ASCII : http ://lmgtfy.com/ ?q=table+ASCII

2.3 Exercice : Désérialiser l'objet

Maintenant que vous avez réussi à obtenir la classe de l'objet sérialisé, vous allez pouvoir le dé-sérialiser. Vous pourrez réutiliser la fonction que vous avez codée dans la première partie du TP (deserializer). Vous devrez tout de même coder une fonction qui créera une variable du bon type, y stockera l'objet dé-sérialisé et renverra cet objet. Vous pourrez ensuite appeler la méthode *display_field ()* pour afficher le contenu des attributs de l'objet. Le fichier binaire ("DeepThought.bin") contenant l'objet sérialisé est situé dans le dossier *bin/debug/* de la solution fournie. En voici le prototype :

```
public static DeepThought deserialize(string filename);
```

La classe que vous avez décryptée contient, en plus des deux champs privés (la clé et la réponse à la question, tout deux cryptés), une fonction display_field () permettant d'afficher le contenu des deux champs, ainsi que des accesseurs et des mutateurs permettant d'y accéder et de les modifier.

2.4 Exercice : MD5 Brute Force

Vous l'aurez peut-être remarqué, l'un des attribut de l'objet contient un hash MD5.

> « *L'algorithme MD5, pour Message Digest 5, est une fonction de hachage cryptographique qui permet d'obtenir l'empreinte numérique d'un fichier (on parle souvent de message).* »
>
> Wikipedia

Autrement dit, pour n'importe quel fichier (ou simplement une string), une fonction de Hash MD5 permet de calculer une "empreinte" de longueur constante qu'il est quasiment impossible d'inverser (c'est à dire qu'il n'est pas possible de retrouver le texte de départ à partir d'un hash). Enfin presque .. il est possible de trouver le message d'origine en testant toutes les possibilités et en comparant à chaque fois le code Hash que l'on souhaite retrouver. Pour cela, nous vous indiquons que le message d'origine a une longueur de **3 caractères** et ne contient que des lettres minuscules.

Afin de vous aider, nous vous fournissons une fonction permettant de calculer le Hash MD5 d'une chaine de caractères (cette fonction est présente dans le fichier Fixme.cs) :

```
using System.Security.Cryptography;
using System.Text;


public static char[] hash (char[] s)
{
  MD5 md5Hash = MD5.Create ();

  byte[] result = md5Hash.ComputeHash (Encoding.UTF8.GetBytes(s));
  StringBuilder sBuilder = new StringBuilder();

  for(int i = 0; i < result.Length; i++)
    sBuilder.Append(result[i].ToString("x2"));

  return sBuilder.ToString();
}
```

Vous allez donc devoir écrire une fonction qui permettra de calculer tous les mots qui sont des combinaisons de 3 caractères. Pour chaque combinaison, vous devrez calculer son Hash MD5 et le comparer au Hash de référence (qui n'est autre que celui dont on veut connaitre l'antécédent par la fonction de Hash). Si vous trouvez le message de départ, vous devrez l'afficher dans la console et le retourner. Voici le prototype :

```
public static char[] brute_hash(char[] hash_ref);
```

Bonus - A moins que ce ne soit déjà fait, créez une deuxième fonction de brute_force qui calcule l'ensemble des combinaisons de 3 caractères **récursivement**. Voici le prototype de cette deuxième fonction :

```
public static char[] brute_hash_rec();
```

A l'issue de cet exercice, vous connaissez normalement le mot dont le code MD5 était stocké dans le premier attribut de l'objet. Passons donc à la dernière étape.

2.5 Exercice : Décryptage à l'aide d'une clé

Voici la dernière étape avant d'accéder à la réponse ! Le message que vous venez d'obtenir en brute forçant le Hash MD5 n'est autre que la clé permettant de décrypter le message contenu dans le deuxième attribut de l'objet que vous avez désérialisé.

Vous parvenez après quelques jours de travail à détecter que le chiffrement utilisé pour crypter le deuxième attribut est un chiffrement de Vigenère.
Le principe de ce chiffrement est sensiblement le même que celui de César : au code ASCII de chaque lettre du message un nombre a été ajouté pour obtenir une nouvelle lettre. Dans le cas du chiffrement de césar, ce nombre est le même pour tout le message. Dans celui du chiffrement de Vigenère, ce nombre est déterminé par la clé que vous avez trouvé.

Cette clé est utilisée de la façon suivante : A chaque lettre est associée une valeur, de 1 à 26 avec a = 1, b = 2, c = 3, etc... Dans le cas d'une clé de deux lettres, la valeur de la **première** lettre de la clé est ajouté à celle de la **première** lettre du message (modulo 26) pour obtenir la valeur de la lettre cryptée. La valeur de la **deuxième** lettre de la clé est ajouté à celle de la **deuxième** lettre du message (modulo 26) puis la clé "boucle" : la **première** lettre de la clé est utilisée pour la **troisième** lettre du message, etc...

Un exemple valant mieux qu'un long discours, voici comment le message "hello world" serait crypté si l'on utilisait la clé "cle" :

```
  'h' | 'e' | 'l' | 'l' | 'o' | ' ' | 'w' | 'o' | 'r' | 'l' | 'd'
   8  |  5  | 12  | 12  | 15  |     | 23  | 15  | 18  | 12  |  4

+ 'c' | 'l' | 'e' | 'c' | 'l' | 'e' | 'c' | 'l' | 'e' | 'c' | 'l'
   3  | 12  |  5  |  3  | 12  |     |  3  | 12  |  5  |  3  | 12
= ---------------------------------------------------------------
  11  | 17  | 17  | 15  |  1  |     | 26  |  1  | 23  | 15  | 16
  'k' | 'q' | 'q' | 'o' | 'a' | ' ' | 'z' | 'a' | 'w' | 'o' | 'p'
```

Le message encodé est donc : "kqqoa zawop"

Nous considèrerons pour cet exercice que les espaces contenus dans le message ne sont pas cryptés et que l'ensemble d'arrivée comme l'ensemble de départ est l'ensemble des caractères minuscules de la table ASCII (pas de caractère accentué, ni de caractère spécial), que ce soit dans la clé, dans le message encodé ou dans le message décodé d'origine.

Voici un pseudo-code d'une fonction capable de crypter un message suivant l'algorithme de Vingenère. Supposez que la fonction char() transforme un nombre entier en caractère de l'alphabet (suivant la règle a = 1, b = 2, c = 3 etc...), que la fonction code() renvoie le code de la lettre donnée et que la fonction len() renvoie la longueur de la chaine donnée.

```
function encode_vingenere (string message, string key)
        for i from 0 to len(message)
                message[i] = char(code(message[i]) +
                                  code(key[i mod len(key)]) mod 26)
        end for
        return message
end function
```

Votre travail : Pour terminer ce TP, et connaître enfin la réponse à la vie, à l'univers et à tout le reste, votre tâche n'est pas de pouvoir chiffrer, mais déchiffrer le message contenu dans le deuxième champs de l'objet. Vous devez pour cela écrire une fonction qui prendra en paramètre le message codé et la clé pour le décoder, et renverra le message décodé. Cette fonction aura le prototype suivant :

```
public static char[] decode_vigenere(char[] msg, char[] key);
```

Ca y est ! Le monde vous acclame, l'univers entier vous respecte, vous avez découvert la réponse à la vie, l'univers et tout le reste !
Votre objectif, désormais, sera de calculer quelle était la question...

... Mais ceci est une autre histoire.