Mise en situation DEV C# : Travail sur les fichiers

A rendre : un programme fonctionnel accessible via un « git » + un manuel technique en format « pdf » contenant le lien vers le « git ».
Il faudra mettre en place un maximum de fonctions et de procédures, sans tomber dans l'excès bien sûr, que ce soit pour l'affichage du menu, le choix de l'option, la saisie des informations sur un client, l'affichage des informations sur un client, l'écriture ou la lecture des informations dans un fichier, etc…. J'en ai déjà trop dit ☺

Objectif : Nous allons mettre en place un programme réalisé en C# qui permet de réaliser des actions sur un fichier « Clients ». Pour chacun d'eux, nous allons enregistrer un numéro de client, le nom, le prénom et le numéro de téléphone. Hormis le numéro de client qui est un entier, chacune de autres informations est de type « Chaine de caractères ».

Il s'agit d'une mise en situation. Je vais donc vous donner les principaux éléments que je veux voir dans le programme final, comme un client pourrait le faire, pour avoir une finalité commune à tous et pouvoir apporter un jugement, et donc une note, mais vous êtes libre dans la manière de coder, de vous organiser, de répondre à la problématique. Attention, le tout doit être tout de même efficace, d'un niveau « bachelor » ! N'oubliez pas les commentaires dans le code, la mise en place de constantes, des noms de variable parlants et bien sûr gérer les cas d'erreurs possibles. Les cas d'erreurs envisagés devront être commentés dans le manuel technique !

A cette étape, nous ne travaillerons pas encore avec la notion de classe pour décrire un client ni la sérialisation pour stocker une instance de classe dans un fichier mais nous allons utiliser la notion de structure (« struct » en C#) et les méthodes des classes techniques permettant de travailler avec les fichiers binaires. Une structure est un moyen comme les classes qui permet de réunir plusieurs propriétés sous une même variable. On n'y place pas obligatoirement de méthodes à l'intérieur comme dans le cadre des classes car les propriétés restent « public ». Il n'y a pas d'instanciation à réaliser contrairement à un objet d'une classe. Pour notre problème, il faudra mettre en place une « struct Client ». Je vous donne vaguement, très vaguement, des indications sur le mode de déclaration et le fonctionnement d'une structure en C#

Voici la syntaxe d'une structure :

```
public  struct   NomStructure {
        public  Type  NomPropriété ;
        public  Type  NomPropriété ;
        …….
        …….
}
```

Déclarer une variable de type « struct »

```
NomStructure  NomVariable ;
```

Pour atteindre un champ dans un programme

```
NomVariable.NomPropriété …..
```

Pour initialiser une variable de type « struct » :

- Vous pouvez déclarer une variable de type « struct » et placer les valeurs souhaitées dans chacun des champs
- Vous pouvez créer unv« constructeur » entre les accolades de la structure puis utiliser la syntaxe pour déclarer la variable et l'initialiser:

  NomVariable=new NomStructure(…….)

Sujet : Le programme à mettre en place va réaliser des actions sur un fichier binaire contenant des fiches « Clients ». Pour chaque client, on mémorise un numéro de client, son nom, son prénom et son numéro de téléphone. Le numéro de client est un entier et le reste est de type « Chaine de caractères ». Au lancement du programme, celui-ci affiche un ensemble d'items de menu et boucle dessus jusqu'à ce que l'utilisateur décide de quitter l'application. A chaque réaffichage du menu, il faudra repartir sur un écran propre donc non pollué de tous les affichages de l'action précédente. L'utilisateur devra sélectionner le numéro de l'item qu'il souhaite réaliser. Il faudra contrôler que le numéro de l'item est possible et que l'action demandée est possible au moment où l'utilisateur la déclenche comme par exemple, je ne peux pas modifier un client si le fichier est vide ☺
Les items définis ci-dessous devront utilisent la notion de fichier binaire et une structure « Client » (cf pensez aussi à mettre en place un « switch ». Voici les items :

1 : Saisir un nouveau client ➔ Le nom du client doit être enregistré en majuscule et le prénom avec la première lettre en majuscule et le reste en minuscule. Vous ajouterez deux procédures de conversion, respectivement de nom « Majuscule » et « FirstMajuscule », recevant en paramètre la chaine à convertir.
2 : Afficher un client ➔ le programme demande à l'utilisateur le nom d'un client puis il recherche celui-ci dans le fichier. S'il n'y a pas de client pour ce nom, nous affichons un message d'erreur. Par contre, s'il y en a plusieurs clients avec le même nom, nous affichons toutes les solutions. Attention, on doit pouvoir saisir le nom en majuscule ou en minuscule même si dans le fichier le nom des clients est en majuscule. Vous ferez également apparaitre le numéro de la fiche ➔ Ce numéro correspond en fait à la position de la fiche dans le fichier
3 : Afficher tous les clients.
4 : Afficher le nombre de clients
5 : Modifier un client ➔ Le programme demande le numéro de la fiche à modifier. Ce numéro correspond en fait à la position de la fiche dans le fichier Si cette fiche existe, le programme affiche les données actuelles concernant la fiche, saisit les nouvelles informations et les enregistre après confirmation de l'utilisateur. Pour saisir, les nouvelles valeurs on peut imaginer que lorsque l'utilisateur ne saisit rien pour l'un des champs, donc fait juste « Entrée », cela signifie que l'on veut conserver la valeur actuelle du champ dans le cas contraire on prend en compte la nouvelle valeur..
6 : Supprimer une fiche. ➔ on réalise une suppression logique ➔ cf paragraphe ci-dessous.
7 : ➔ cf paragraphe ci-dessous
8 : ➔ cf paragraphe ci-dessous
9 : ➔ cf paragraphe ci-dessous
10 : Quitter.

Remarque :

- Pour supprimer une fiche, on devra réaliser une suppression logique en mettant par exemple un astérisque dans le champ « Nom » de la fiche supprimée. Dans ce cadre, lorsqu'on ajoute une nouvelle fiche, il faudra d'abord regarder s'il n'y a pas une place à récupérer dans le fichier avant d'ajouter la nouvelle fiche à la fin. On pourra également prévoir trois nouvelles options dans le menu qui permettent pour la première de compresser le fichier en supprimant physiquement toutes les fiches marquées comme étant supprimées logiquement (option 9), pour la deuxième d'afficher uniquement les fiches marquées comme supprimées logiquement (option 8) et pour la troisième de récupérer une fiche supprimée par erreur (option 7). Dans ce dernier cas, il faudra demander à l'utilisateur de taper le nom du client qui viendra donc remplacer l'astérisque dans le champ « Nom » et donc fera que la fiche ne sera plus considérée comme étant supprimée logiquement.
- L'option 3 : affiche les fiches non supprimées, donc existantes, pas toutes les fiches. Remarque : on fera apparaitre également le numéro de la fiche devant chaque fiche. Le numéro de la fiche correspond à la position de la fiche dans le fichier.
- L'option 4 : affiche le nombre de fiches non supprimées, donc existantes, pas le nombre total de fiches. Par contre on peut ajouter un item « Stat » qui affiche le nombre de fiches total, le nombre de fiches en suppression logique et autres infos utiles !
- Pour compresser le fichier, il faut éliminer les fiches marquées comme étant supprimées. Pour réaliser cela, vous créerez un fichier temporaire dans lequel vous copierez les fiches non supprimées se trouvant dans le fichier de base. Ensuite, il faut

supprimer le fichier de base puis renommer le fichier temporaire avec le nom du fichier de base.

- Il faut également prendre en compte le fait que lors de la modification d'informations, les infos modifiées n'ont peut-être pas la même taille que les infos déjà en place ☺ A vous de trouver une solution
- Il serait bien que chaque item de menu soit associé à l'appel d'une procédure qui réalise l'action. Dans quel but me dirait vous ? Eh bien, je souhaite rendre dynamique le menu et avoir tous les textes des items dans une liste dynamique. Cela me permettra, si je veux ajouter des actions supplémentaires dans mon menu, de simplement ajouter le texte dans la liste des items et ne pas devoir insérer le texte dans le menu si celui-ci est statique. Cette solution de menu dynamique a aussi pour conséquence de créer une liste des noms de procédures associées à des items de menu. Chaque numéro d'item dans le menu correspondra à sa position dans la liste des items mais également au même numéro dans le liste des noms de procédures associés aux items. Il suffira alors que je récupère le nom de la procédure dans la liste des procédures à partir du numéro sélectionné par l'utilisateur dans le menu et que je déclenche cette procédure. Voir le code exemple ci-dessous pour voir comment on peut appeler une procédure via une liste dynamique de noms de procédure
  Remarque, il est également possible d'utiliser la notion de « Dictionnaire » en C#. A vous de découvrir son fonctionnement et l'intérêt car il utilise un système de couple « Clé, Valeur » qui permettrait de ne pas avoir à gérer deux listes en parallèle (La liste des items et la liste des noms de procédures) mais une seule liste composée du couple « Texte Item / Nom Procédure »

Exemple de code permettant d'appeler une procédure à partir d'une chaine stockée dans une liste sachant que dans notre cas, les procédures associées aux items de menu n'ont pas de paramètre à gérer ce qui simplifie la mise en place

```
using System;
using System.Collections.Generic;
using System.Reflection; // Contient le nécessaire pour déclencher la procédure à partir de son nom

// Mise en place du programme principal
public class MonProg {

   // Liste des procédures (méthodes)
   public void Methode1(string Valeur){
      Console.WriteLine($"Méthode 1 : {Valeur}");
   }

   public void Methode2(){
      Console.WriteLine($"Méthode 2");
   }

   public void Methode3(){
      Console.WriteLine($"Méthode 3");
   }

   // Programme principal
   public static void Main() {
      // Liste des noms de méthodes
      List<string> LesMethodes = new List<string> { "Methode1", "Methode2", "Methode3" };
      bool Ok=true;

      // Instance de la classe contenant les méthodes à déclencher
      MonProg monprog = new MonProg();

      Console.WriteLine("Entrez le numéro de la méthode souhaitée");
      int Num=int.Parse(Console.ReadLine());
      // Utilisation de reflection pour obtenir la méthode par son nom
      MethodInfo methodInfo = monprog.GetType().GetMethod(LesMethodes[Num-1]);
```

```
        // Vérifier si la méthode existe
        if (methodInfo != null)
           {
              // Appeler la méthode mais contrôler la présence de paramètre
               // Vérifie si la méthode attend des paramètres
              ParameterInfo[] parameters = methodInfo.GetParameters();
              if (parameters.Length == 0)
              {
                 // Appeler la méthode sans paramètre
                 methodInfo.Invoke(monprog, null);
              }
              else
              {
                 // Appeler la méthode avec des paramètres
                 // On va placer les paramètres dans une liste de la classe "object" ce qui permet
                 // d'y placer des valeurs de n'importe quel type
                 List<object> LesArgs = new List<object>();
                 Console.WriteLine($"La méthode attend {parameters.Length} paramètre(s)");
                 foreach (ParameterInfo param in parameters)
                 {
                    Console.WriteLine($"- Nom paramètre : {param.Name}, Type paramètre : {param.ParameterType}");
                    // Demande la saisie d'une information du type du paramètre
                    switch (Type.GetTypeCode(param.ParameterType))
                     {
                        case TypeCode.Int32:
                              int intResult;
                              Console.WriteLine("Entrez une valeur de type Entier : ");
                              int.TryParse(Console.ReadLine(), out intResult);
                              LesArgs.Add(intResult);
                              break;
                        case TypeCode.String:
                              Console.WriteLine("Entrez une valeur de type string : ");
                              string input=Console.ReadLine();
                              LesArgs.Add(input);
                              break;
                        case TypeCode.Double:
                              double doubleResult;
                              Console.WriteLine("Entrez une valeur de type double : ");
                              double.TryParse(Console.ReadLine(), out doubleResult);
                              LesArgs.Add(doubleResult);
                              break;
                        default:
                              Console.WriteLine("Type non pris en charge");
                              Ok=false;
                              break;
                       }
                 }
                 if (Ok) {
                    Console.WriteLine("Appel de la méthode demandée");
                    methodInfo.Invoke(monprog, LesArgs.ToArray());
                 }
                 else
                    Console.WriteLine("Appel de la méthode demandée impossible");
              }

           }
     }
}
```