Programmation objet Classe EnsCar

Vous êtes chargé de mettre en place une classe permettant de limiter les caractères autorisés pour un champ de saisie. Cette classe repose sur la gestion d'un tableau dynamique (une collection ou List) contenant les caractères autorisés. Les méthodes vont permettre de gérer cette collection.

Pour cela la classe suivante doit être mise en place :

```
public class EnsCar {
    List<char> lesCaracteres;  // Collection (tableau dynamique) de Caractères

    public EnsCar() {…} // instanciation de la collection
    public void ajouter(char c ) {…}      // ajoute c à la collection lesCaracteres s'il n'est pas déjà présent
    public bool retirer(char c )  {…}     //  retire c de la collection lesCaracteres si il est présent
    public bool contenir(char c) {…}      // renvoie true si la valeur de c est présente dans la collection
    public void vider() {…} // supprime tous les caractères de la collection
    public bool estVide() {…}  // retourne vrai si la collection est vide
    public string getLesCaracteres() {…}
        // retourne dans une chaîne les caractères présents dans la collection
}
```

Une collection est un tableau dynamique qui offre des méthodes pour ajouter, retirer, rechercher et trier L'exemple ci-dessous permet de manipuler une collection de caractères. Le principe est le même quel que soit le type des éléments.

Déclaration d'une collection de caractères	List<char> lesCaracteres;
Instanciation de la collection	lesCaracteres = new List<char>();
Ajout d'un caractère à la collection	lesCaracteres.**Add**('e');

Parcours de la collection	for (char unCaractere in lesCaracteres) {
	  Console.WrileLine(unCaratere);
	}

Accès au premier caractère de la collection	lesCaracteres[0]
Nombre de caractères dans la liste	lesCaracteres.**Count**
Tester si la liste est vide	if (lesCaracteres.Count == 0) {…}
Tester si un caractère est présent dans la liste	if (lesCaracteres.Contains('d') {…}
Tester si un caractère n'est pas présent	if (!lesCaracteres.Contains('d') {…}
Vider la collection	lesCaracteres.**clear();**
Trier la collection	lesCaracteres.**Sort()**

La classe EnsCar contient donc un tableau dynamique de caractères (Collection) qui contiendra les caractères autorisés.

La solution fournie comporte actuellement 2 projets :

Enscar	**Bibliothèque de classe** (.NET Framework 4.8) contenant la définition de la classe EnsCar à compléter
TestUnitaireEnscar	**Projet de test unitaire** (.NET Framework 4.8) permettant de tester les différentes méthodes de la classe Enscar. Certains tests sont fournis (encapsulés dans un commentaire), d'autres sont à réaliser.

Le squelette de la solution sous Visual Studio vous est fourni. Elle contient un projet de bibliothèque de classes et un projet de test unitaire permettant de vérifier le bon fonctionnement des méthodes.

01	Compléter le fichier ensCar.cs définissant la classe EnsCar.
02	Vérifier à l'aide du projet de test unitaire le bon fonctionnement de vos méthodes en lançant les tests suivants : TestMethodeAjouter1() - TestMethodesSupprimer1() – TestMethodeAppartient() Attention : afin d'éviter les erreurs de compilation, chaque test est placé dans un commentaire, il faut donc décommenter le test pour pouvoir le lancer.
03	Ajouter à la classe Enscar une méthode intersection(ens : EnsCar) : EnsCar qui retourne un objet Enscar contenant les caractères communs entre l'objet appelant (this) et l'objet passé en paramètre (ens). Réaliser une version statique de cette méthode : intersection(ens1 : EnsCar, ens2 : Enscar) : Enscar
04	Vérifier à l'aide du projet de test unitaire le bon fonctionnement de vos méthodes en lançant le test unitaire testMethodeIntersection
05	Ajouter à la classe Enscar une méthode difference(ens : EnsCar) : EnsCar qui retourne un objet Enscar contenant les caractères présents dans l'objet appelant (this) mais pas dans l'objet passé en paramètre (ens). Réaliser une version statique de cette méthode : difference(ens1 : EnsCar, ens2 : Enscar) : Enscar
06	Vérifier à l'aide du projet de test unitaire le bon fonctionnement de vos méthodes en lançant le test unitaire TestMethodeDifference()

L'initialisation d'un objet de la classe EnsCar s'avère peu pratique car les méthodes 'ajouter' et 'retirer' ne permettent de travailler que sur un seul caractère. Il serait intéressant de surcharger ces deux méthodes en prenant une chaine de caractères en paramètre :

```
Procédure EnsCar.ajouter( liste : chaîne)
    c : caractère
    Pour Chaque c dans liste Faire
            lesCaracteres.ajouter(c)
    Fin Pour
```

```
Procédure EnsCar.supprimer( liste : chaîne)
    c : caractère
    Pour Chaque c dans liste Faire
        lesCaracteres.supprimer(c)
    Fin Pour
```

07	Surcharger les méthodes ajouter et retirer pour permettre d'ajouter ou de supprimer les caractères contenus dans une chaîne passée en paramètre.
08	Compléter la méthode de test TestMethodeAjouter2 qui réalise le même test que la méthode TestMethodeAjouter1 mais en utilisant la méthode ajouter surchargée. Vérifier le bon fonctionnement du test.
09	Compléter la méthode de test TestMethodeSupprimer2 qui : Crée un objet unEnsemble contenant les lettres "abcdef" Supprime "ace" et vérifier qu'il reste "bde" Supprime "bdf" et vérifier que l'objet est vide. Vérifier le bon fonctionnement du test.

En C# il est possible de définir des procédures ou des fonctions avec un nombre variable de paramètres.
Syntaxe : void ajouter(params char[] liste)
La boucle foreach s'utilise de la même façon pour récupérer la valeur de chaque paramètre.

Avec ce principe on peut surcharger les méthodes ajouter et supprimer pour qu'il soit possible d'ajouter plusieurs caractères en une seule fois par un appel de la forme ens1.ajouter('a', 'c', 'e', 'f')

10	Surcharger de nouveau les méthodes ajouter et retirer pour permettre d'ajouter ou de supprimer les caractères passés en paramètre.

11	Contrôler ces deux nouvelles méthodes en ajoutant dans le projet de test unitaire les méthodes TestMethodeAjouter3 et TestMethodeSupprimer3 (sur le modèle de la version 2)

Nous allons appliquer ces deux surcharges au niveau du constructeur. Cela va permettre de remplir un objet de la classe dès son instanciation.

12	Surcharger le constructeur de la classe pour qu'il admette les surcharges suivantes : EnsCar ensemble1 = new EnsCar('g', 's', 'f', 'a', 't', 'p', 't', 'c', 'r'); EnsCar ensemble2 = new EnsCar("gsfatprcr");
13	Contrôler ces deux nouveaux constructeurs en ajoutant dans le projet de test unitaire la méthode TestConstructeur qui : Créer un objet ensemble à l'aide du constructeur par défaut et vérifier que l'objet est vide. Créer l'objet ensemble1 et vérifier qu'il contient "acfgprst" Créer l'objet ensemble2 et vérifier qu'il contient "acfgprst"

La classe est maintenant totalement définie et opérationnelle. Elle peut être générée sous la forme d'une DLL qui pourra être utilisée dans des applications.

Pour générer la dll : Clic droit sur le projet EnsCar – Générer. La dll Enscar.dll est générée dans le répertoire bin/debug du projet.

Notre objectif est maintenant de mettre en place une application permettant la saisie d'un nom et le contrôle de ce nom en utilisant un objet de la classe Enscar qui contient les caractères autorisés.

L'objet EnsCar sera initialisé avec les caractères autorisés dans un nom :

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz '-"
```

Nous allons réaliser 3 versions de cette application, une version en mode console, et 2 versions graphiques(WinForms et WPF).

Pour éviter de réécrire la même partie du code dans ces 3 applications, il faut **séparer la logique métie**r (validation du nom) de l'interface utilisateur (console ou interface graphique).

Pour cela nous pouvons encapsuler la logique métier dans une classe statique possédant une méthode statique estValide(string nom)

```
namespace Sio {
   public static class CtrlNom     {
      private static readonly EnsCar lesCaracteresAutorises = new
EnsCar("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz '-");

      public static bool estValide(string nom)        {
         EnsCar lesCaracteresSaisis = new EnsCar(nom);
         EnsCar lesCaracteresNonAutorises = EnsCar.difference(lesCaracteresSaisis,
lesCaracteresAutorises);
         return lesCaracteresNonAutorises.estVide();
      }
   }
}
```

14	Ajouter cette classe (ctrlnom.cs) dans le projet de type 'bibliothèque de classe'. Générer la solution afin d'obtenir la dll 'enscar.dll' .

Il ne reste plus qu'à utiliser cette dll dans nos trois applications. Commençons par la version en mode console.

15	Réaliser dans une nouvelle solution 'testEnscar', un projet en mode console 'testConsole' permettant de saisir un nom et de contrôler sa validité à l'aide d'un objet de la classe CtrlNom qui contient les caractères autorisés (les lettres, l'espace, l'apostrophe et le tiret). Le programme doit demander la saisie du nom et le faire jusqu'à ce que ce dernier soit valide. Répeter Saisir nom Si Non CrtlNom.estValide(nom) Alors Afficher "Ce nom n'est pas validen veuillez réessayer" Fin Si Tant que Non CrtlNom.estValide(nom) Afficher "Ce nom est valide" Afficher "Appuyez sur une touche pour fermer la fenêtre…" ReadKey Pour pouvoir utiliser la dll enscar.dll qui contient les classes Enscar et CtrlNom : Ajouter au niveau du projet une référence vers la dll Enscar.dll : Clic droit sur le projet – Ajouter – Référence – Onglet Parcourir – Bouton Parcourir Ajouter la clause 'using Sio;' au début du programme (les classes ont été définies dans cette espace de nom)

Pour réaliser une application utilisant une interface graphique, deux technologies sont utilisables aussi bien dans le ".NET Framework 4.8" (Windows C#7.3) que dans le Framework ".NET 9.0" (multiplateforme C#13)

Windows Forms (WinForms) est une technologie **ancienne mais toujours utilisée** pour créer des interfaces graphiques sous Windows uniquement.

- Introduit avec .NET Framework 1.0 (début des années 2000)
- Utilise des **contrôles graphiques classiques** comme boutons, labels, boîtes de texte
- L'interface se conçoit via un **designer visuel** (glisser-déposer)
- Utilise le moteur graphique GDI/GDI+ sans accélération matérielle
- L'interface est **gérée en pixel**, donc moins souple pour des interfaces modernes ou responsives

Avantages :

- Très simple à apprendre et à mettre en place
- Rapide pour des petites applis internes ou outils simples
- Bien intégré à Visual Studio

Inconvénients :

- Limitée en style graphique (pas d'effet visuels, animation limitée, pas très responsive)
- Pas optimisé pour la séparation logique
- Vieille technologie (mais encore supportée)

16	Créer dans une nouvelle solution 'testWF' un projet de type Application Windows Forms (.NET Framework)' nommé 'testWF'. La solution et le projet seront placés dans le même répertoire  Pour pouvoir utiliser la dll enscar.dll qui contient les classes Enscar et CtrlNom : Ajouter au niveau du projet une référence vers la dll Enscar.dll : Clic droit sur le projet – Ajouter – Référence – Onglet Parcourir – Bouton Parcourir Ajouter la clause 'using Sio;' au début du programme (les classes ont été définies dans cette espace de nom)

WPF (Windows Presentation Foundation) est une technologie plus **moderne**, introduite avec .NET 3.0 (2006), pour créer des interfaces graphiques **riches et personnalisables**.

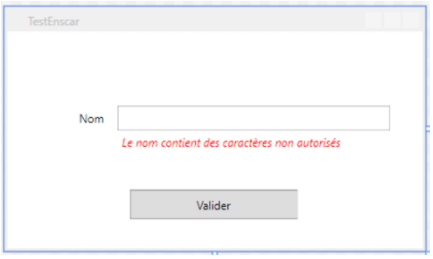
- Utilise **XAML** (un langage déclaratif) pour décrire l'interface
- Utilise le moteur graphique DirectX qui offre le rendu 2D/3D, les effets graphique et l'accélération GPU
- Supporte **les animations, les styles, les templates**, la 3D, le binding de données
- Conçu autour du modèle **MVVM (Model-View-ViewModel)** pour une architecture propre

Avantages :

- Plus souple pour créer des interfaces responsives et complexes
- optimisé pour la séparation logique
- Binding puissant entre données et interface
- Bon support pour les animations, transitions, vecteurs

Inconvénients :

- Courbe d'apprentissage plus longue
- Plus complexe pour des petits projets
- Moins intuitif pour les débutants si on ne connaît pas le XAML

17	Créer dans une nouvelle solution "testWPF" un projet de type 'Application WPF (.NET Framework)' nommé 'testWPF' réalisant le même travail.  Pour pouvoir utiliser la dll enscar.dll qui contient les classes Enscar et CtrlNom : Ajouter au niveau du projet une référence vers la dll Enscar.dll : Clic droit sur le projet – Ajouter – Référence – Onglet Parcourir – Bouton Parcourir Ajouter la clause 'using Sio;' au début du programme (les classes ont été définies dans cette espace de nom)