Application Gestion Garage

Version : 1.0

L'objectif est de créer une mini application permettant la gestion d'un garage.

A rendre un fichier compressé (zip, 7z, rar) contenant :

- Le projet Visual Studio contenant l'ensemble des fichiers de la solution et du projet.

Vous pouvez réaliser cette application par groupe de deux.

Nommer le fichier nom.prenom.zip ou nom1.nom2.zip et mettre celui-ci dans le dossier de rendu MonCampus prévu à cet effet ou envoyer le fichier à l'adresse **nicolas.chevalier@reseau-cd.net** .

Première étape :

Objectifs :

- Développer une application en utilisation les notions avancées de POO
- Mettre en place les notions de classe et méthode abstract
- Mettre en place les notions d'interface

On désire développer une application permettant de gérer le parc de véhicules qu'un garage pourra proposer à la vente. L'application devra permettre de gérer aussi bien des voitures, que des camions et des motos.

Une voiture sera caractérisée par son nom, son prix, sa marque, un moteur, des options, une puissance en chevaux fiscaux, un nombre de porte, un nombre de siège et une taille de coffre (exprimé en m3).

Un camion sera caractérisé par son nom, son prix, sa marque, un moteur, des options, un nombre d'essieux, un poids de chargement et un volume de chargement.

Une moto sera caractérisée par son nom, son prix, sa marque, un moteur, des options et une cylindrée.

Pour chaque véhicule, on devra pouvoir effectuer les actions suivantes :

- Afficher l'ensemble des informations d'un véhicule.
- Ajouter une option au véhicule
- Connaitre la marque du véhicule
- Connaitre le moteur du véhicule
- Connaitre les options du véhicule
- Connaitre le prix du véhicule (hors options et taxe)
- Calculer la taxe sur le véhicule
- Connaitre le prix total du véhicule

Chaque véhicule disposera également d'un « Id » qui sera automatiquement généré à partir de 1. Vous pouvez utiliser pour cela une variable static permettant de faire la génération des Id avec une incrémentation de cette variable à chaque ajout d'un véhicule.

La taxe sur les véhicules se calcule de la manière suivante :

- Pour une voiture, on multiplie la puissance en chevaux fiscaux par 10 €,
- Pour un camion, on prend le nombre d'essieux et on multiplie par 50 €,

- Pour une moto, on multiplie la cylindrée par 0,3 euros et on prendra la partie entière.

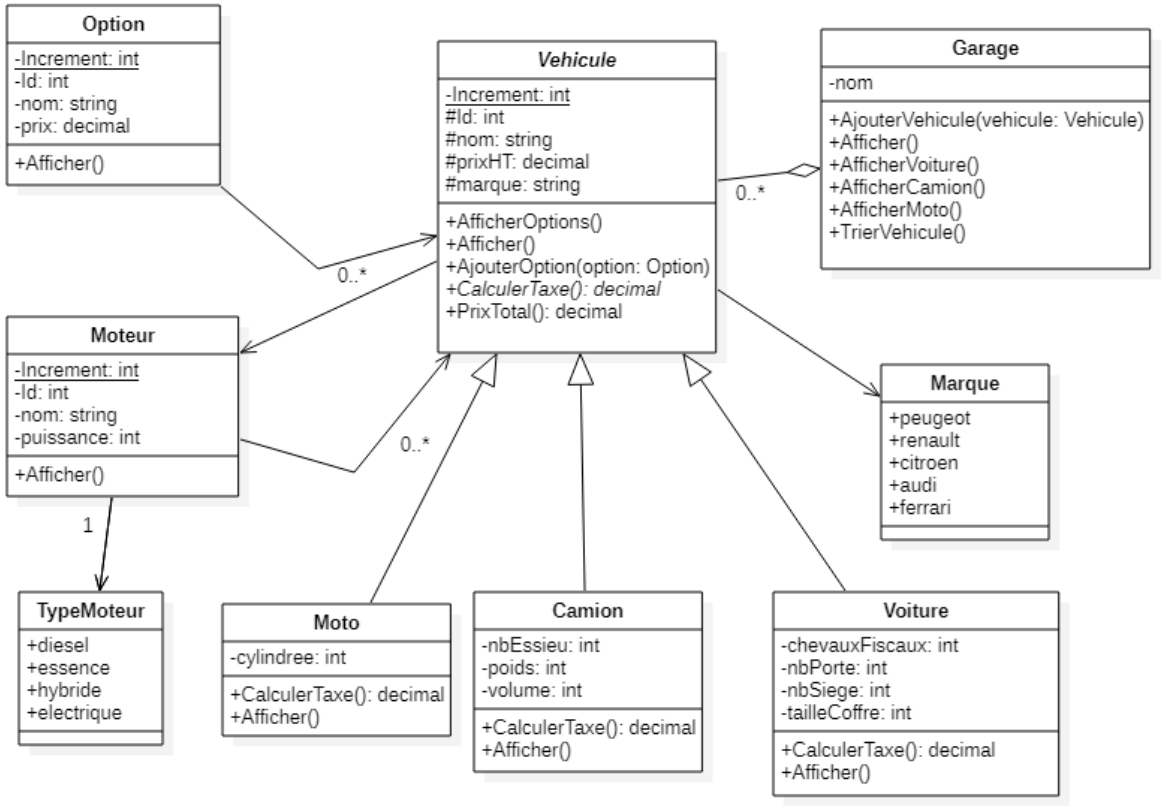
Un moteur sera caractérisé par un type (diesel, essence, hybride, électrique) et d'une puissance. On devra pouvoir afficher l'ensemble des informations d'un moteur. Un moteur disposera également d'un id avec le même système que pour un véhicule.

Une option sera caractérisée par un nom et un prix. On devra pouvoir afficher l'ensemble des informations d'une option. . Une option disposera également d'un id avec le même système que pour un véhicule.

On devra pouvoir ajouter aux garages des véhicules, trier les véhicules suivant leur prix, et afficher l'ensemble des informations concernant les véhicules du garage.

Le tri sera mis en place avec l'implémentation de la classe IComparable du Framework .Net. Vous trouverez un exemple d'implémentation de cette classe dans le support de cours « Programmation orientée objet C# Notions avancées ».

Modélisation UML du Garage :



Travail à effectuer :

1. Développer l'ensemble des classes de l'application
2. Créer un jeu de tests de véhicules afin de les ajouter au garage et afficher ensuite tous les véhicules de ce garage. Tester également le tri et afficher les véhicules une fois trié.

Deuxième étape :

A partir de la première version de l'application de gestion d'un garage, nous allons ajouter des fonctionnalités supplémentaires permettant d'ajouter dynamiquement des véhicules via un menu, et de sauvegarder ces informations dans un fichier.

Objectifs :
- Utiliser les exceptions
- Créer nos propres exceptions
- Traiter des données saisies
- Effectuer des opérations sur des listes
- Utiliser le mécanisme de sérialisation pour enregistrer des informations dans un fichier

Afin de mieux gérer le garage et de pouvoir enregistrer les informations de celui-ci dans un fichier, nous allons réaliser un menu permettant d'effectuer les commandes suivantes :

1. Afficher les véhicules
2. Ajouter un véhicule
3. Supprimer un véhicule
4. Sélectionner un véhicule
5. Afficher les options d'un véhicule
6. Ajouter des options à un véhicule
7. Supprimer des options à un véhicule
8. Afficher les options
9. Afficher les marques
10. Afficher les types de moteurs
11. Charger le garage
12. Sauvegarder le garage
13. Quitter l'application

Une classe Menu permettra de mettre en place l'affichage du menu et toutes les commandes.

Afin de gérer le garage cette classe aura comme constructeur :

```
public Menu(Garage garage)
{
        this.garage = garage;
}
```

Il s'agira du garage créé dans le programme principal.

Afin de stocker également des moteurs et des options, il faudra rajouter dans votre garage une liste de moteurs, une liste d'options et les méthodes AjouterMoteur et AjouterOption.

Une méthode Start() permettra de lancer le menu à partir du programme. Cette méthode affichera le menu, et bouclera tanque l'utilisateur n'aura pas sélectionné le choix 13 pour sortir de l'application.

Pour traiter les exceptions de l'application on mettra un bloc try … catch global autour du menu qui permettra de catcher les Exceptions levées par les commandes du menu et d'afficher un message d'erreur à l'utilisateur.

Dans cette méthode Start() un switch case permettra de tester les valeurs saisie par l'utilisateur. Cette valeur étant récupérée sous forme de string il faudra convertir cette valeur en int afin de la tester dans le switch case.

Dans la classe menu créer une méthode GetChoix() qui permettra :

- Récupérer le choix saisi par l'utilisateur
- Effectuer la conversion en int
- Retourner le résultat converti

Cette méthode devra implémenter une première gestion d'exception pour gérer le cas où l'utilisateur ne saisit pas un nombre. L'exception levée lors de la conversion est la FormatException. Il faudra dans cette méthode catcher l'exception puis lever à nouveau la FormatException avec le message d'erreur : "Le choix saisie n'est pas un nombre".

Le message sera affiché par le bloc try … catch de la méthode Start.

Il faudra aussi afficher un message si l'utilisateur saisie un choix qui n'existe pas dans le menu.

Pour cela nous allons créer une Exception spécifique à notre application.
Créer cette Exception dans le fichier Menu.cs et nommer la MenuException. Vous trouverez comment créer une nouvelle exception personnalisée dans le fichier « Programmation orientée objet C-Sharp Exeptions.pdf ».
Utiliser le constructeur de l'exception pour spécifier un message qui pourra être "Le choix n'est pas compris entre 0 et 11".

Créer une méthode GetChoixMenu() qui utilisera la méthode GetChoix() pour récupérer le nombre saisie et qui vérifiera si le nombre est compris entre 0 et 11. Si ce n'est pas le cas il faudra lever l'exception MenuException.

Le message sera affiché par le bloc try … catch de la méthode Start.

Pour chaque opération du menu créer une méthode, qui sera chargée d'effectuer toutes les opérations nécessaires de la commande.

Par exemple pour la commande « 1. Afficher les véhicules », on créera une méthode :

```
public void AfficherVehicules()
{

}
```

Qui sera appelé dans le switch case :

```
case 1:
        AfficherVehicules();
        break;
```

Créer toutes les méthodes correspondant à chaque fonction prévue dans le menu.

Coder toutes les fonctionnalités en essayant de mettre dans la classe Garage les fonctions de base et dans la classe Menu l'interface avec l'utilisateur.

Ajouter également de nouvelles exceptions pour traiter par exemple les cas où l'utilisateur sélectionne un véhicule qui n'existe pas, une marque qui n'existe pas, un moteur qui n'existe pas, …

Dernière étape :

Modifier les différentes classes Vehicule, Voiture, Camion, … afin d'ajouter le mécanisme de sérialisation.

il suffit d'ajouter l'attribut :

[Serializable]

A chaque classe.

Ajouter en vous inspirant du support « La sérialisation des objets.pdf » dans votre classe Garage les deux méthodes :

Enregistrer : Permet d'enregistrer un objet.
Charger : Permet de charger un objet.

Ajouter dans votre gestion de menu et dans votre programme la fonction de sauvegarde et de chargement d'un garage.