LATIF Wail

DECHANE Sanaâ

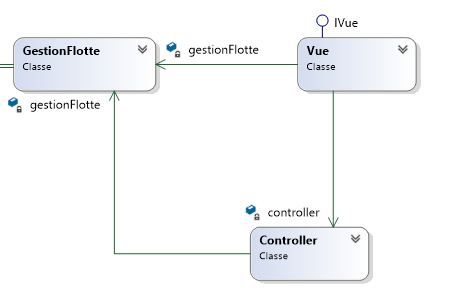
Rapport programmation orienté objet avancée projet

C# : Logiciel de gestion de flotte

Tout d’abord nous avons choisi de structurer notre code en suivant le modèle MVC(Modèle Vue Contrôleur).

Nous avons donc dans le package MVC une classe Vue qui permet à l’utilisateur de communiquer avec les classes métier, elle ne fait aucun traitement et contient seulement des fonctions d’affichage et de saisie. Cette Vue implémente l’interface IVue qui nous permet de définir plusieurs vue différentes (Console,Graphique …) qui ont toutes en commun la fonction Start() qui démarre l’interface. La classe Vue est associé à la classe GestionFlotte afin d’avoir accès aux données pour les afficher mais aussi à la classe Controller qui est dans le même package et qui permet de valider la saisie de l‘utilisateur.

Le schéma d’utilisation est donc simple la vue permet l’accès au donnée ainsi que la saisie de donnée le Controlleur va ensuite valider les saisie et mettre a jour le Modèle (class GestionFlotte) .



Concernant les packages Nous avons décidé de mettre en place un package :

* Modèle qui contient toutes les classes métiers
* MVC contenant les classe vue et Controller
* Interfaces
* Exceptions
* Utils contenant les class utilitaire utilisée dans le programme

# Class :

GestionFlotte : class Maitresse

Herite : aucune

Implémente : Sauvegardable

Attribue :

* List de vehicule,
* List de client
* List de trajet
* List de parking
* 3 contrôleur ( à ne pas confondre avec Controller)

Tous ces attributs sont en readonly donc initialisable uniquement dans le constructeur afin de bien encapsuler les List et contrôleur car le contenue des listes peut changer mais pas leur instance

Méthode :

AjoutVehicule() : permet d’ajouter un véhicule et de lui attribuer une place disponible dans l’un des parking

GetParkingsDisp() : retourne une list contenant les parking avec des places disponibles

GetPlaceDisp() : Retourne une place disponible dans l’un des parking du gestionnaire de flotte

Vehicule :

Classe abstraite non initialisable ;

Attributs : - un numéro de véhicule afin de l’identifier (on a créé un attribut statique lastNVehicule qui permet de définir des numéros de véhicules unique)

- une marque

- un modèle

- une couleur

- un kilomètrage total(km)

- un attribut IsDisponible afin de savoir le véhicule est disponible ou non

- un coût qui sera calculé via une méthode abstraite dépendant du type de véhicule

- place dans laquelle est stationné la voiture (null si la voiture est en cours de location)

- un numéro de trajet auquel est associé le véhicule s’il n’est pas disponible et =-1 sinon

- une liste contenant les interventions effectuées sur le véhicule par le contrôleur.

Nous avons volontairement omis l’attribue immatriculation car inutile dans notre modèle

Methode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = ""): ecrit dans le fichier via le fWriter l’objet en question en format json.
* Abstract CalculerCout() : calcule le cout du vehicule suivant une formule.
* AddIntervention(Intervention intervention) : ajout une intervention à la list
* DelLastIntervention() : permettant de supprimer la dernière intervention sur le véhicule car il est impossible de supprimer une intervention si ce n’est la dernière pour éviter toute fraude.

Pour les différents types de véhicules, nous avons 3 classes : Moto, Voiture et Camion.

Ces trois classes ont en commun les attributs de la classe Vehicule dont ils héritent.

Pour chaque véhicule on a décidé d’encapsuler les attributs le plus possible, c’est pourquoi par exemple la liste d’intervention est en private et n’a pas de propriétés.

*ClassMoto :*

Hérite : Vehicule

Implémente : Sauvegardable

Attribue :

* Cylindre (cm3) compris entre 50 et 1500

Methode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = "")
* CalculerCout() : Coût de la moto= cylindré x 0,20

*ClassCamion :*

Hérite : Vehicule

Implémente : Sauvegardable

Attribue :

* capacite (L) compris entre 2,75 et 22L

Methode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = "")
* CalculerCout() : Coût = capacité x 37,5 (37,5 euros le litre)

Remarque : la consommation d’une moto et d’un camion est plus grande que celle d’une voiture et leur usage mécanique est plus important.

*ClassVoiture :*

Hérite : Vehicule

Implémente : Sauvegardable

Attribue :

* NbPorte compris entre 3 et 5
* Puissance compris entre 70 et 650 chevaux
* Type de voiture (break berline ou monospace)

Methode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = "")
* CalculerCout() :
  + Break : cout=puissance
  + Berline : cout=puissance x1,5
  + Monospace : cout=puissance x1,25

Car le coefficient dépend du cout d’entretien des différents types de voiture

*ClassClient:*

Hérite : Aucune

Implémente : ISauvegardable

Attribue :

* Readonly NClient : numero du client readonly car ne change pas
* NbAnneePermis : nombre d’année des permis
* Nom
* Prenom
* Adresse
* PermisList : liste des permis
* TotalLoc : location total en euro
* Static LastNClient : numéro du dernier client

Méthode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = "")

*ClassTrajet:*

Hérite : Aucune

Implémente : ISauvegardable

Attribue :

* Readonly nTrajet : une fois le trajet cree son numéro ne doit plus changer
* Readonly client : une fois le trajet cree le client ne doit plus changer
* Readonly vehicule : une fois le trajet cree le vehicule associé ne doit plus changer
* Distance
* date
* Cout

Méthode :

* Sauvegarder(StreamWriter fWriter, string before = "", string after = "")
* Supprimer() : permet de reduire le kmtrage du vehicule associer au trajet ainsi que totalloc du client associé au trajet lorsque le trajet est supprimé
* CalculerCout() :
  + voiture : cout=cout du Vehicule + 0,25\*distance
  + reste : cout=cout du Vehicule + 0,5\*distance

*ClassParking:*

Hérite : Aucune

Implémente : Aucune

Attribue :

* readonly Place[] places : tableau de 10 place en readonly pour les meme raison que precedemment
* int NbPlaces = 10 : nombre de place dans le parking non constant car un parking peut être agrandit
* nom : nom du parking

Méthode :

* GetPlace(int numPlace) : retourne la place correspondant au numero
* GetPlaceDisp(): retourne une place libre null si aucune ne l’est
* GetPlacesDisp(): retourne une liste de places libre null si aucune ne l’est

*ClassVue:*

Hérite : Aucune

Implémente : IVue

Attribue :

* controller
* gestionFlotte

Méthode :

* Start() : la seul methode public de cette class elle permet de démarrer l’interface
* Les méthodes d’affichage de menu : affichage d’un menu et appel les méthodes relatives au choix de l’utilisateur géré avec un switch case
* Les méthodes de saisie (client, trajet …) : saisie des paramètres et les envoie au Controller pour vérification si des exception son lever on appel la méthode EndFunction pour réexécuter la méthode ou pour afficher le menu d’accueil
* Sauvegarder() : crée un StreamWriter avec comme parametre le path du fichier de sauvegarde et pour chacun véhicule, client, trajet des liste de gestionFlotte elle appel la fonction sauvegarder afin de sauvegarder l’état du gestionnaire en json
* Charger() : on récupère toute les donnée du fichier de sauvegarde en format string et on les convertit en JObject (JsonObject du packet .netJson) afind de faciliter leur traitement puis pour chaque list vehicule, trajet et client elle appelle la methode du controlleur qui leur est liée ChargerVehicules, ChargerClients, ChargerTrajets
* EndFunction(string msg, Action After = null, string before = null) : permet d’afficher un msg du genre "Veuillez appuyez sur une touche pour continuer ..." d’attendre la saisir d’une touche du clavier puis d’appeler la fonction After si non null et si before n’est pas null il sera afficher avec le msg