Projet C++ : Covoiturage

Brandon Goumba, Lucie Grousset, Cheikh Cisse
Master 2 MAPI3
Année universitaire 2022/2023

1 Introduction

Pour ce projet, nous avons décidé de prendre comme sujet le covoiturage.

Pour se faire nous avons découpé le travail en plusieurs parties.
Une partie pour les personnes (clients, conducteurs), une suivante pour le paiement par carte, une pour le paiement par paylib, une partie véhicule, ensuite une pour le trajet, une pour la date, une autre pour l'horaire, une pour la ville, ainsi qu'une dernière pour l'application.

La classe personne, représente une classe mère. Ce qui est spécifié entre parenthèse représente les classes filles associées à cette classe mère.

Dans la partie suivante, nous expliquerons plus en détail la définition de chacune de ces classes.
Avant cela nous allons brièvement expliquer en quoi consiste notre projet.

C'est une application pour du covoiturage, il consiste à prendre en compte un conducteur et un client qui veulent partager un même trajet.
Le trajet dépend de la ville de départ, celle d'arrivée puis de l'horaire et de la date. En fonction du trajet, du véhicule et du nombre de passagers, le client devra payer son conducteur une certaine sommes, définie par l'application en fonction de ces paramètres. Pour le paiement, le client possède deux possibilités, par carte ou par paylib.
Arrivé à destination, le conducteur et le client peuvent noter le trajet effectué.

2 Classes

2.1 Classe personnes

Nous allons tout d'abord voir la définition de la classe mère (classe personnes).
Définition de la classe personnes :
- Attributs : nom, email et numéro de téléphone.
- Constructeur : vide et avec paramètres.
- Méthode : surcharge de l'opérateur de sortie « : afficher une personne.
- Méthode : méthode virtuelle satisfaction (renvoie 1 si le client est satisfait ou 0 sinon).

Maintenant nous observons les classes filles et nous commençons par la classe client:
Définition de la classe client :
- Attributs : numéro de ticket.
- Constructeur : vide et avec paramètres.
- Méthodes : affiche_client (qui affiche toutes les informations du client), annuler_voyage (qui demande à un client s'il souhaite oui ou non annuler son trajet) et satisfaction (renvoie 1 si le client est satisfait ou 0 sinon).

Nous regardons ensuite la classe fille, classe conducteur:
Définition de la classe conducteur :
- Attributs : identité, permis de conduire, véhicule.
- Constructeur : vide et avec paramètres.
- Méthodes : affiche_conducteur (afficher toutes les données du conducteur), accepter (test du nombre de place pour accepter ou refuser le client) et satisfaction (renvoie 1 si le client est satisfait ou 0 sinon).

2.2 Classe paiement par carte

Définition de la classe paiement par carte :
- Attributs : numéro de la carte, date de validité et cryptogramme(code).
- Constructeur : vide et avec paramètres.
- Méthodes : affiche_carte (affiche tous les attributs de la carte) et payer (permet d'effectuer le paiement).

2.3 Classe paiement par paylib

Définition de la classe Paiement par paylib:

- Attributs : numéro de téléphone.
- Constructeur : vide et avec paramètres.
- Méthodes : affiche_number (affiche le numéro de téléphone) et payer (permet d'effectuer le paiement).

2.4 Classe véhicule

Définition de la classe véhicule :
- Attributs : type, état, immatriculation, nombre de places et nombre maximum de places.
- Constructeur: vide et avec paramètres.
- Opérateur : surcharge de l'opérateur de sortie « qui permet d'afficher les attributs du véhicule.
- Méthodes : test_nplaces (renvoie vrai ou faux selon si le véhicule possède assez de place ou non).

2.5 Classe Trajet

Définition de la classe trajet :
- Attributs : ville d'origine, ville destination, date, heure.
- Constructeur: vide et avec paramètres.
- Opérateur == : permet de savoir s'il n'y a pas de trajet identique qui existe.
- Méthodes : affiche_trajet (permet d'afficher le trajet à effectuer), saisir_trajet (permet à l'utilisateur de saisir les informations sur son trajet).

2.6 Classe Date

Définition de la classe Date :
- Attributs : jour, mois, année
- Opérateur == : test pour savoir si deux dates sont identiques.
- Méthodes : TestAnneeBiss (permet de tester si c'est une année bissextile), TestDate (permet de tester si la date est valide), affiche_date (permet d'afficher la date (jour, mois, années)) et saisir_date (permet de saisir la date (jour, mois, années).

2.7 Classe Horaire

Définition de la classe Horaire :
- Attributs : heure départ, heure arrivée.
- Opérateur == : test pour savoir si des horaires sont identiques.
- Méthodes : affiche_horaire (permet d'afficher l'heure), saisir_horaire (permet de saisir l'heure et les minutes).

2.8 Classe ville

Définition de la classe ville :
- Attributs : nom, identifiant de la ville.
- Constructeur: vide et avec paramètres.
- Opérateur == : permet de vérifier qu'il n'y est pas deux villes identiques .
- Méthodes : affiche_ville (permet d'afficher le nom de la ville), saisir_ville (permet d'entrer le nom de la ville).

2.9 Classe Application

Définition de la classe Application :
- Attributs : Nombre maximum de client, nombre de client, nombre de conducteur, nombre maximum de conducteur.
- Constructeur : vide et avec paramètres.
- Méthodes : recherche_personnes (nous permet de trouver tous les clients et tous les conducteurs pour un certain trajet).

3 Conclusion

Notre projet nous permet donc de trouver un trajet commun à au moins un client et un conducteur.

Le système global du projet est fait par interface. Nous avons une première interface avec deux choix, le premier choix est voyager et le deuxième choix est afficher les trajets et les personnes sur ce trajet. Si on choisi l'option 1, ceci nous amène sur une deuxième interface donc à faire un deuxième choix, cette fois ci entre client, conducteur et un test de satisfaction. Si nous choisissons au contraire l'option deux nous obtenons l'affichage du réseau. Ainsi de suite jusqu'à arriver à notre objectif final qu'un conducteur puisse prendre au moins un client pour un trajet défini.

Nous remarquons toutefois que plusieurs points pourraient être à améliorer dans ce projet. On peut dire que ce projet n'est pas réel, par exemple pour le paiement on n'utilise pas une carte mais seulement des chiffres, pour le nombre de trajet nous nous sommes contenté d'en faire 4 différents et sans arrêt possible, un trajet partira d'un unique point A vers un unique point B pour chaque passager.