Initiation à C++

Exercices Projet

Août 2016

Nicolas Benoit

Le Concessionnaire

Introduction

Le but final de cet exercice est de créer un empire commercial basé sur tout ce qui a un moteur. L'entreprise possèdera plusieurs concessions qui pourront faire preuve de souplesse dans leurs tarifs pour attirer le chaland. Mais pour commencer il faudra développer plusieurs types de véhicules et une unique concession.

N'hésitez surtout pas à développer vos propres évolutions ou à personnaliser le projet si vous avez des idées.

Ce projet est à développer en TDD

Des moteurs

- Créez une classe Véhicule contenant comme variables de classe trois String : m_marque, m_modele et un int : m_prix. Ajoutez un constructeur, une méthode description() renvoyant une string et une méthode changerPrix(int nouveauPrix).

- Créez une classe Carrosserie contenant comme variables trois string : m_couleur, m_matiere et m_finission et une méthode description(). Intégrez aux Vehicules un attribut carrosserie, le constructeur de Vehicule doit également créer sa Carrosserie et la méthode description() de Véhicule doit également appeler la description() de Carrosserie.

- Ajoutez une méthode améliorerCarrosserie() à la classe Véhicule. L'appel à cette méthode modifie sa carrosserie pour que m_matiere = "Fibre Carbone" et m_finission = "Perlee". SI les deux variables contenaient un texte différent, le prix est doublé, si une des variable contenait un texte différent, le prix augmente de 50%.

- Créez maintenant deux classes héritant de Vehicule : Moto et Voiture. Chacune a sa nouvelle variable de classe. La Moto un int cylindree et la Voiture un String type (citadine, coupé, limousine...). Ecrivez leur méthode description() en vous servant de la description() de la classe mère avec un ajout relatif à leur variable propre.

Des Concessions

- Créez maintenant le nouvel objet Concession avec comme variables une String m_nom et un vecteur de shared pointers sur Vehicules. Créez ensuite la méthode ajouterVehicule(shared_ptr<Vehicule> nouveauVehicule) qui ajoute un Vehicule dans le vecteur.

- Ajoutez maintenant à votre Concession la méthode inventaire() qui appelle la méthode description() de chacun des Vehicules. Prenez soin de vérifier que le polymorphisme est respecté.

- Ajouter maintenant la méthode calculerValeurStock() qui retourne la somme cumulée des prix des Vehicules dans la Concession.

- Ajoutez à votre Concession la variable de classe trésorerie initialisée dans le constructeur. Ecrivez maintenant la méthode vendre(int i) qui vend le ième véhicule, c'est à dire que cette méthode ajoute le prix du Véhicule à la trésorerie mais l'enlève du vecteur de véhicules. Gérez le cas où vous rentreriez un i qui ne corresponde à rien dans le vecteur.

- Au lieu de simplement ajouter un Vehicule dans une concession, celle-ci doit maintenant l'acheter. Ecrivez la méthode acheter(Vehicule) qui soustrait à la trésorerie de la concession 80% du prix du Vehicule (il faut bien qu'elle fasse de la marge !) et l'ajoute à son vecteur.

- Ajoutez la méthode reduction(int pourcentage) qui réduit le prix de tous les Véhicules de la Concession du pourcentage passé en paramètre (reduction(20) réduit les prix de 20%). De même, créez la fonction augmentation(int pourcentage) pour profiter d'une opportunité de marché.

- En pratique, nous n'allons jamais avoir dans notre concession un objet de type Vehicule, il s'agit d'une entité théorique destinée à l'héritage. Faire de cette classe une classe abstraite.

- Ajoutez la méthode liquidationMarque(String marque) qui vend tous les véhicules d'une certaine marque.

Un Empire

- Lassé d'avoir une simple concession ? Créez une classe EmpireMotorise qui prend comme variable de classe un tableau de Concessions et qui implémente des méthodes faisant appel aux méthodes des Concession sur toute la liste (réduction sur un type de véhicule, liquidationMarque, inventaire...). Il sera également possible de vendre une concession ce qui permettra de récupérer ses liquidités.

- Implémentez la méthode deplacerStock(Concession aVider, Concession aRemplir) qui transfère les véhicules d'une concession à une autre. Surchargez cette méthode pour qu'elle ne transfère que les véhicules d'une certaine marque

Mise en Application des Concepts du Cours

UML

Réalisez le schéma UML représentant les classes mises en jeu.

La Surcharge d'Opérateurs

- Surchargez les opérateurs d'égalité et d'inégalité (== et !=) de Vehicule. Servez vous en pour ajouter une méthode à vos concessions retournant le nombre de véhicules différents qu'elle contient.

- Surchargez l'opérateur de comparaison < de vos Concessions et de vos Vehicules, la concession considérée comme la plus grande étant celle qui a la valeur de stock la plus grande et le Vehicule le plus grand étant le plus cher. Cela servira plus tard pour appliquer un algorithme de tri.

Pile et Tas

Etes vous certain que votre programme ne génère aucune fuite mémoire ?

Strings

Pour ceux qui aiment jouer avec les strings, créez un générateur de plaque d'immatriculation.

Pour satisfaire la norme française (deux lettres - trois chiffres - deux lettres) faîtes ainsi :

Sélectionnez deux lettres au hasard dans la marque pour les 2 premières lettres, de même avec le modèle pour les 2 dernières lettres (assurez vous de ne prendre que des lettres) et faites un random de 1 au prix du véhicule le tout modulo 1000 pour les trois chiffres.

Assurez vous que votre plaque d'immatriculation est bien unique.

Foncteurs et Algorithmes

Si vous avez sauté la partie sur la surcharge d'opérateurs il est temps d'y revenir ! Nous avons besoin d'une surcharge de l'opérateur <

Codez pour vos Concessions et Vehicules un foncteur de tri.

Ajoutez à votre Concession la méthode trierVehicules et à votre Empire la méthode trierConcessions qui effectuent le tri à l'aide de l'algorithme sort.

Exceptions

Avez vous pensé au cas où une concession achèterait un Vehicule sans avoir l'argent ? Si oui c'est très bien, si non c'est le moment de vous rattraper !

- Faites en sorte qu'acheter un Véhicule sans avoir l'argent nécessaire déclenche une exception.

Unique_ptr

Envie de défi ?

Remplacez tous les shared_ptr de votre programme par des unique_ptr.

C'est presque pareil... Sauf que vous n'avez plus le droit de faire la moindre copie, même temporaire !

Pensez à la fonction move() ;)