INF1010

Programmation Orientée-Objet

Travail pratique #1
Allocation dynamique, composition et agrégation

Objectifs : Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation.

Remise du travail : Mardi 17 septembre 2019, 8h00 (AM).

Références : Notes de cours sur Moodle & Chapitre 2-7 du livre Big C++ 2e éd.

Documents à remettre : Les fichiers .cpp et .h **uniquement**, complétés et réunis sous la forme d'une archive au format **.zip**.

Directives : Directives de remise des Travaux pratiques sur Moodle
Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.
Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. **Pas de remise possible sans être dans un groupe.**
Veuillez suivre le guide de codage

La directive de précompilation « #ifndef »

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'entête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entrainerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H

#define NOMCLASSE_H

// Définir la classe NomClasse ici

#endif
```

La directive de précompilation « #include»

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

Mise en contexte

L'entreprise Air Poly, une compagnie de transport aérien à Montréal, décide de lancer son propre programme de fidélité dans l'espoir de développer des clients loyaux. L'objectif est de proposer un système de points aux utilisateurs. Plus les clients achètent chez Air Poly, plus ils ont de points. Ils peuvent ensuite échanger ces points contre des coupons rabais.

Cependant, l'entreprise a besoin d'aide pour mettre en place son système de gestion du programme de fidélité, où leurs administrateurs pourront gérer les billets, membres etc. C'est ici que vous intervenez.

Travail à réaliser

On vous demande de compléter les fichiers qui vous sont fournis pour pouvoir implémenter le système décrit ci-dessus.

Les fichiers billet.h, coupon.h, membre.h, gestionnaire.h et main.cpp vous sont fournis. On vous demande de compléter les fichiers *.cpp en suivant les indications qui vont suivre.

Note : Attention à bien lire les spécifications générales plus bas.

Classe Coupon

Cette classe caractérise un coupon rabais par son code, son taux de rabais et son coût en points.

Cette classe contient les attributs privés suivants :

- code_ (string) .
- rabais_ (double) : montant du rabais, en pourcentage. Sa valeur est comprise entre 0 et 1.
- cout_ (int) : coût du coupon, en points.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut de votre choix.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modification.
- Une méthode d'affichage (voir exemple d'affichage).

Classe Billet

Cette classe représente un billet d'avion. Un billet est tout le temps associé à un membre.

Elle contient les attributs privés suivants :

- pnr_ (string) : numéro de réservation du billet. Est utilisé comme un ID.
- nomPassager_ (string) : nom du passager à qui est attribué le billet.
- prix_ (double) : prix du billet, après rabais.
- od_ (string) : origine - destination. Représente l'itinéraire du trajet avec l'aéroport de départ et de destination.
- tarif_ : tarif du billet (économie, affaire…).
- dateVol_ (string) : date du vol associé au billet.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut de votre choix.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications.
- formatTarif : converti le type de tarif tiré de l'enum TarifBillet (voir fichier def.h) à la chaîne de caractère correspondante.
- Une méthode d'affichage (voir exemple d'affichage).

Classe Membre

Cette classe caractérise un membre du programme. Un membre est en tout temps associé au gestionnaire du programme.

Elle contient les attributs privés suivants :

- nom_ (string).
- points_ (int).
- billets_ (Billet**), tableau des billets d'avions associés à ce membre.
- nbBillets_ (int) : nombre de billets que possède actuellement le membre.
- capaciteBillets_ (int) : capacité du tableau de billets.
- coupons_ (Coupon**) : tableau des coupons que possède ce membre. Un membre peut avoir plusieurs fois le même coupon.
- nbCoupons_ (int) : nombre de coupons que possède actuellement le membre.
- capaciteCoupons_ (int) : capacité du tableau de coupons.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut de votre choix. Pour la capacité des tableaux, utilisez la constante CAPACITE_INITIALE définie dans le fichier def.h.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes. La capacité est initialisée de la même façon que dans le constructeur par défaut.
- Les méthodes d'accès.
- Les méthodes de modification.
- ModifierPoints : ajoute le nombre de points reçu en paramètre au total de points du membre. La valeur reçue en paramètre peut être négative.
- AjouterBillet : crée et ajoute un billet au membre à partir des paramètres reçus. Modifie aussi le nombre de points du membre. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un billet.
- AjouterCoupon: ajoute un coupon au membre. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un coupon.
- RetirerCoupon : supprime la première occurrence du coupon dans le tableau de coupon. Il ne doit pas y avoir de trou au milieu du tableau après la suppression, mais avoir une valeur ignorée au bout du tableau est accepté.
- AcheterCoupon : ajoute le coupon au tableau de coupons si le membre possède assez de points. Modifie le nombre de points que possède le membre.
- CalculerPoints : calcule le nombre de point que rapporte un billet. Le nombre de points que rapporte un billet est égal à 10% de son prix. Additionnellement, dépendamment du tarif du billet, le billet peut rapporter des points bonus
    - Premium économie : 50 points
    - Affaire : 150 points
    - Première : 300 points
- Une méthode d'affichage (voir exemple d'affichage).

Classe Gestionnaire

Cette classe représente le cœur du programme. Elle permet de gérer les différents membres, la liste des coupons disponibles et de réaliser diverses autres opérations.

Elle contient les attributs privés suivants :

- membres_ (Membre**) : tableau des membres du programme.
- nbMembres_ (int) : nombre de membres actuellement géré par le gestionnaire.
- capacitéMembres (int) : capacité du tableau de membres.

- coupons_ (Coupon**) : tableau de tous les coupons disponibles dans le programme. Il contient donc tous les coupons auxquels peuvent avoir droit les membres. A noter qu'il diffère du tableau de coupons de la classe membre, qui représente seulement les coupons que possède un membre en particulier et qui peut contenir des duplicats.
- nbCoupons_ (int) : nombre de coupons actuellement dans le gestionnaire.
- capaciteCoupons (int) : capacité du tableau de coupons.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. Utilisez la constante CAPACITE_INITIALE comme pour celui de la classe membre.
- Les méthodes d'accès.
- AjouterMembre : crée et ajoute un membre au programme. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un membre.
- AjouterCoupon : crée et ajoute un coupon au programme. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un coupon.
- TrouverMembre : cherche un membre dans la liste de membre en fonction de son nom. Si aucun membre est trouvé, un nullptr est retourné et un message d'erreur est affiché
- AssignerBillet : assigne un billet à un membre. Si un coupon est utilisé, calcule le prix réel après avoir utilisé le meilleur coupon possible, avant de l'assigner à un membre.
- AppliquerCoupon : calcule et utilise le meilleur coupon que possède le membre. Pour cela, elle cherche dans la liste des coupons du client celui qui a le plus grand taux de réduction, puis renvoie le montant du rabais sur le prix. Si le membre ne possède aucun coupon, un message d'erreur est affiché.
- AcheterCoupon : permet d'acheter un coupon pour un client. C'est toujours le coupon avec le plus grand rabais possible qui est acheté. Si le membre n'a pas assez de point pour acheter un coupon, un message d'erreur est affiché.
- Une méthode d'affichage (voir exemple d'affichage)

Main.cpp

Le fichier main.cpp vous est fourni et ne devrait pas avoir à être modifié pour la remise. N'hésitez pas à commenter certaines parties si vous avez besoin de compiler votre code. Vous pouvez aussi utiliser ce fichier pour mieux comprendre les requis des fonctions.

Une fois tous les fichiers complétés, tous les tests devraient passer.

Fichiers complémentaires

En plus des fichiers ci-dessus, l'énoncé contient le fichier def.h qui contient les constantes utilisées dans le programme, et le fichier debogageMemoire.hpp. Ce dernier est présent pour vous aider à vérifier s'il y a des fuites de mémoire dans votre code. Exécutez la solution en débogage pour qu'il fonctionne. Si une fuite de mémoire est détectée, un message sera affiché dans la fenêtre Sortie (Output) de Visual studio.

Exemple d'affichage

Voici un exemple d'affichage du programme que vous pouvez utiliser pour vous inspirer. Pour votre travail, vous devez afficher les mêmes informations. Cependant, vous êtes libres au niveau de la forme et de l'esthétique.

```
Le membre ccc n'existe pas
Le membre ne peut acheter de coupon
Le membre n'a pas de coupon utilisable
TESTS
        Test 01... OK!
        Test 02... OK!
        Test 03... OK!
        Test 04... OK!
        Test 05... OK!
        Test 06... OK!
        Test 07... OK!
        Test 08... OK!
        Test 09... OK!
        Test 10... OK!
        Test 11... OK!
        Test 12... OK!
        Test 13... OK!
        Test 14... OK!
        Test 15... OK!
        Test 16... OK!
        Test 17... OK!
        Test 18... OK!
        Test 19... OK!
        Test 20... OK!
        Test 21... OK!
        Test 22... OK!
        Test 23... OK!
        Test 24... OK!
        Test 25... OK!
        Test 26... OK!

================== ETAT ACTUEL DU PROGRAMME ==================

- Membre Marc:
        - Points  : 1960
        - Billets :
                - Billet D4E5F6 (Classe : Affaire)
                        - Passager : Marc
                        - Prix     : 2000$
                        - Trajet   : YUL - YYZ
                        - Vol le   : 2019-12-21
                - Billet H7I8J9 (Classe : Affaire)
                        - Passager : Marc
                        - Prix     : 1600$
                        - Trajet   : YUL - YYZ
                        - Vol le   : 2019-12-21
        - Coupons :
                - Coupon 10%. Rabais : 0.1.

- Membre John:
        - Points  : 150
        - Billets :
                - Billet A1B2C3 (Classe : Premium economie)
                        - Passager : John
                        - Prix     : 1000$
                        - Trajet   : YUL - YYZ
                        - Vol le   : 2019-12-21
        - Coupons :

- Membre Robert:
        - Points  : 0
        - Billets :
        - Coupons :
```

Spécifications générales

- Ajoutez un destructeur pour chaque classe lorsque nécessaire
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajoutez le mot-clé *const* chaque fois que cela est pertinent.
- Ne modifiez pas les signatures des méthodes, sauf pour ajouter le mot-clé *const*
- Appliquez un affichage « user friendly » (ergonomique et joli) pour le rendu final.
- Documentez votre code source.
- Ne remettez que les fichiers .h et .cpp lors de votre remise.
- **Bien lire le barème de correction ci-dessous.**

Questions

1. Quel est le lien (agrégation ou composition) entre les classes Membre et Coupon?
2. Quel effet aura une méthode si elle a un const ?

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme
- (4 points) Exécution du programme
- (4 points) Comportement exact des méthodes du programme
- (2 points) Documentation du code et bonne norme de codage
- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés
- (1 points) Utilisation adéquate des directives de précompilation (#ifndef)
- (2 points) Allocation et désallocation appropriées de la mémoire
- (2 points) Réponse aux questions