

INF1010
Programmation Orientée-Objet

Automne 2019

Travail pratique #3
Héritage

Objectifs :	Permettre à l'étudiant de se familiariser avec l'héritage et la conversion de type.
Remise du travail :	Mardi 8 octobre 2019, 8h (AM)
Références :	Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.
Documents à remettre :	Tous les fichiers .cpp et .h exclusivement complétés et réunis sous la forme d'une archive au format .zip.
Directives :	<u>Directives de remise des Travaux pratiques sur Moodle</u> 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. <u>Veuillez suivre le guide de codage</u>

Travail à réaliser

Le travail consiste à continuer l'application pour le programme de fidélité d'air Poly commencée aux TP précédents en y intégrant les notions d'héritage et de conversion de type.

Afin de diversifier leurs offres, Air Poly ajoute des billets de type Flightpass qui représente un ensemble de billets que l'on peut utiliser quand on veut (en plus de l'avantage d'avoir le même prix pour chaque billet). De cette façon, nous avons maintenant des billets réguliers avec une date de départ et des Flightpass qui dérivent les deux d'une classe de base Billet. De plus, afin d'améliorer le programme de fidélité, des membres premium et des membre réguliers ont été ajoutés. Le membre de base, qui est un membre occasionnel, a été modifié pour avoir moins d'avantages. Les membres réguliers ont accès aux coupons alors que les membres premium ont davantage de bonus comme des rabais sur le prix des billets et le prix de coupon. Le type de membre régulier dérive donc de la classe membre alors que les membre premiums eux vont dériver de la classe membre régulier.

L'héritage permet de créer une classe de base possédant ses propres caractéristiques et de l'utiliser pour créer de nouvelles classes. Les nouvelles (dérivées) classes ont avec la classe de base, la relation « est un » et partagent avec la classe de base certaines caractéristiques. Pour réussir ce TP il faudra vous familiariser avec la fonction C++ **static_cast<T*>(Objet*)**. Cet opérateur permet de convertir à la compilation un pointeur d'une classe en un pointeur d'une autre classe. Dans ce TP, il vous permettra par exemple de convertir un pointeur de type **Membre** en pointeur de type **MembrePremium** ou **MembreRegulier**.

Par exemple :

```
Membre* membre = new MembrePremium();
```

```
MembrePremium* membrePremium = static_cast<MembrePremium*>(membre);
```

Cela convertie le pointeur de membre vers un pointer de membre Premium.

Pour vous aider, les fichiers corrigés du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas et les méthodes à modifier vous ont été indiquées.

ATTENTION :

- Pensez à modifier les méthodes d'affichage déjà écrites pour qu'elles s'adaptent aux nouvelles classes.
- Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.
- Vous serez pénalisés pour les utilisations inutiles du mot-clé this. Utilisez-le seulement où nécessaire.
- Il faut utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP2.
- Écrire les destructeurs lorsque c'est nécessaire.

Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h et à l'énoncé.

Classe Coupon

Cette classe caractérise un coupon rabais par son code, son taux de rabais et son coût en points. Il n'y a rien à changer dans cette classe.

Classe Billet

Cette classe représente un billet d'avion. Un billet est tout le temps associé à un membre. Son attribut dateVol_ a été déplacé dans la classe BilletRégulier.

Elle contient l'attribut privé suivant :

- typeBillet_; c'est le type du billet. Régulier, Billet de base ou FlightPass.

Les méthodes suivantes doivent être implémentées :

- Les méthodes d'accès et de modification de l'attribut typeBillet.

Les méthodes suivantes doivent être modifiées :

- L'opérateur << pour l'adapter au retrait de l'attribut dateVol_.

Classe BilletRegulier

Cette classe caractérise un billet avec une date de vol contrairement au FlightPass. Cette classe dérive de la classe Billet.

Elle contient l'attribut privé suivant :

- dateVol_; c'est la date du vol.

Les méthodes suivantes doivent être implémentées :

- Le constructeur par paramètres initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès et de modification de l'attribut dateVol_.
- L'opérateur << qui affiche les caractéristiques du Billet régulier. **(Penser à la réutilisation)**

Classe FlightPass

Cette classe représente un ensemble de 10 billets d'avions que l'on peut acheter d'un coup. Il n'y a pas de date de vol car ceux-ci peuvent être utilisés n'importe quand. Elle dérive de la classe Billet.

Elle contient l'attribut privé suivant :

- nbUtilisationsRestante_; est le nombre d'utilisations restantes du Flightpass, à chaque utilisation il décrémente de 1. **Cet attribut doit être initialisé à la constante NB_UTILISATION_INITIALE.**

Les méthodes suivantes doivent être implémentées :

- Le constructeur par paramètres initialise les attributs aux valeurs correspondantes.

- La méthode d'accès de l'attribut nbUtilisationsRestante.
- La méthode decrementeNbUtilisations qui décrémente de 1 l'attribut nbUtilisationRestante.
- L'opérateur << qui affiche les caractéristiques du FlightPass (voir l'affichage à la fin de l'énoncé).

Classe Membre

Les attributs et les méthodes reliées au programme de fidélité (points et coupons) ont été déplacées dans la classe Membre régulier. Un membre est en tout temps associé au gestionnaire du programme.

Elle contient l'attribut privé suivant :

- typeMembre; cet attribut représente le type du membre, soit un Membre_Occasionnel, soit un Membre_Regulier, soit un Membre_Premium.

La méthode suivante doit être implémentée:

- La méthode utiliserBillet ; cette méthode utilise le billet envoyé en paramètre. Ce billet doit être trouvé dans le tableau de billet du membre puis, il doit être supprimer. Si le billet est un FlightPass, vous devez décrémente le nombre d'utilisations restantes. S'il reste 0 utilisations, le FlightPass doit être retiré des billets que le membre possède. Sinon, le FlightPass ne doit pas être supprimé. Si le billet n'est pas du type FlightPass, il doit tout simplement être supprimé du vecteur de billets sans faire les vérifications précédentes. La méthode affiche aussi un message d'erreur si le billet n'est pas trouvé avec le pnr passé en paramètre.

Les méthodes suivantes doivent être modifiées:

- Le constructeur par défaut et par paramètres a changé pour inclure le nouvel attribut (typeMembre_).
- ajouterBillet; Cette méthode doit créer le bon type de billet selon le type de billet passé en paramètre avant de l'ajouter au vecteur de billet.
- Le constructeur de copie de membre doit créer le bon type de billet (selon typeBillet) avant de l'ajouter à son tableau.
- L'opérateur << qui affiche les caractéristiques du Membre (**Attention** afficher le billet selon le type du billet) (voir l'affichage à la fin de l'énoncé).

Classe MembreRegulier

Il s'agit d'un membre abonné au programme de fidélité. Il peut accumuler des points avec des achats de billet, puis il peut utiliser ces points pour acheter des coupons. (**Pensez à la réutilisation du code dans la classe de base**)

Les méthodes suivantes doivent être implémentées:

- Le constructeur par paramètres initialise les attributs aux valeurs correspondantes;

- L'opérateur qui affiche les caractéristiques du Membre Régulier (voir l'affichage à la fin de l'énoncé).
- ajouterBillet; Cette méthode doit créer le bon type de billet selon le type de billet passé en paramètre ET modifier les points (contrairement à Membre).

Classe MembrePremium

Cette classe dérive de la classe MembreRégulier. Elle contient le nombre de points total accumulés depuis l'adhésion du membre (incluant les points utilisés). Cela permet de donner de meilleurs rabais ! **Attention, pensez à la réutilisation du code des classes parents.**

Elle contient les attributs privés suivants :

- joursRestants_ : le nombre de jour restant à l'abonnement premium. Cet attribut doit être initialisé à la constante JOUR_RESTANT_INITIALE.
- pointsCumules_ : le nombre de points total gagnés depuis le début de l'adhésion du membre en tant que premium. Cet attribut doit être initialisé à 0.

Les méthodes suivantes doivent être implémentées:

- Le constructeur par paramètres initialise les attributs aux valeurs correspondantes (**utilisez la constante JOUR_RESTANT_INITIALE**).
- Les méthodes d'accès et de modification des attributs dont la signature est dans le fichier membrePremium.h.
- La méthode modifierPointsCumules additionne le nombre de points cumulés avec la valeur en paramètre.
- La méthode acheterCoupon qui permet d'acheter un coupon mais avec un rabais qui augmente de 0.01 (1%) par 1000 point cumulé jusqu'à occurrence de 20% de rabais maximum. nbr points, /1000, *1%
- La méthode ajouterBillet qui doit aussi modifier le nombre de points cumulés.
- L'opérateur << qui affiche les caractéristiques du Membre Premium (voir l'affichage à la fin de l'énoncé).

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. **Attention, utilisez les méthodes de la classe approprié. (Pensez à utiliser static cast)**

Les méthodes suivantes doivent être modifiées:

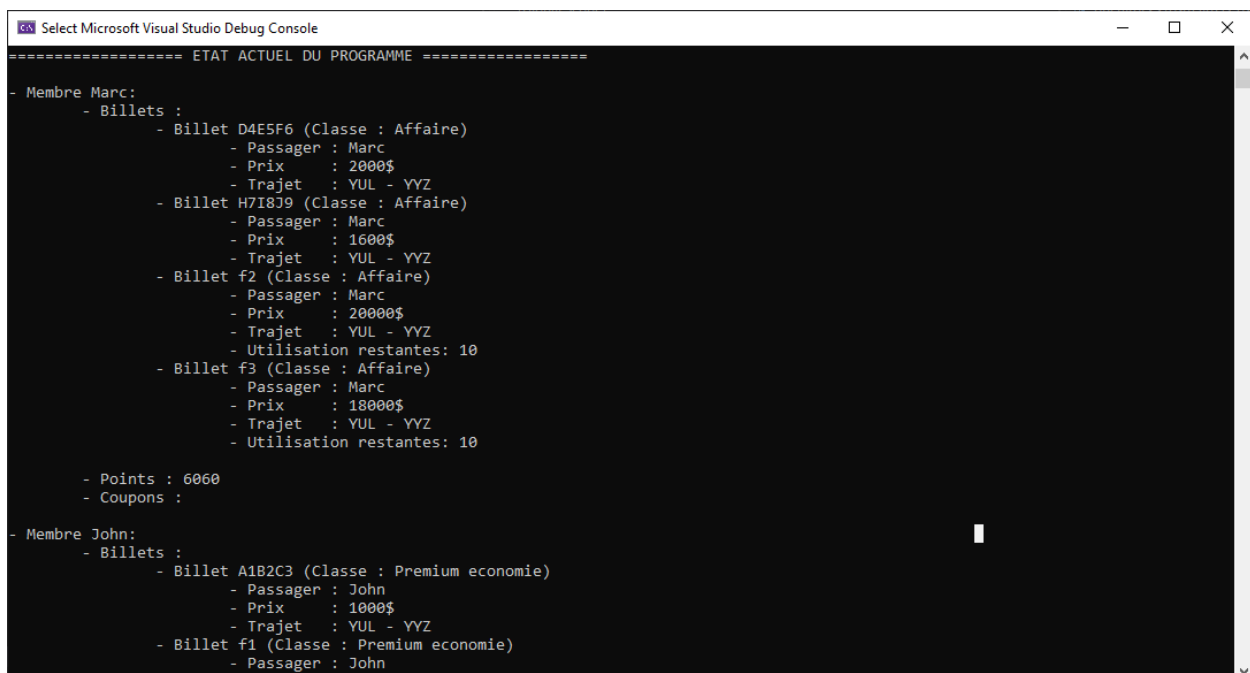
- La méthode assignerBillet remplit la même fonction que le TP précédent. Cependant, on doit changer le prix du billet selon le type de billet. Si le Billet est un FlightPass, le prix

doit être multiplié par 10 car les FlightPass possèdent 10 utilisations. Ensuite, si le Membre est un membre Premium, le prix du billet doit être réduit de 0.005 (0.5%) par 1000 points cumulés du membre premium jusqu'à occurrence de 10% de rabais. Faites attention à appeler les bonnes méthodes de la bonne classe (pensez à static cast) afin de bien appliquer les points selon le type de membre.

- La méthode acheterCoupon doit être modifiée afin d'être adaptée aux membres premiums qui ont un rabais sur le cout des coupons de 0.01 (1%) par 1000 points cumulés jusqu'à occurrence de 20% de rabais.
- La méthode appliquerCoupon doit être modifiée car les membre Occasionnel (classe Membre) n'ont pas de coupons.
- La méthode ajouterMembre doit créer et ajouter le bon type de membre selon le type en paramètre.
- L'opérateur << qui affiche les caractéristiques du gestionnaire doit être adaptée selon le type de membre (voir l'affichage à la fin de l'énoncé).

Main.cpp

Le fichier main.cpp vous est fourni et ne doit pas être modifié pour la remise, à l'exception de l'ajout des réponses aux questions. 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. De plus, votre affichage devrait avoir une apparence semblable à l'exemple de la prochaine page.



```
===== ETAT ACTUEL DU PROGRAMME =====
- Membre Marc:
  - Billets :
    - Billet D4E5F6 (Classe : Affaire)
      - Passager : Marc
      - Prix : 2000$
      - Trajet : YUL - YYZ
    - Billet H7I8J9 (Classe : Affaire)
      - Passager : Marc
      - Prix : 1600$
      - Trajet : YUL - YYZ
    - Billet f2 (Classe : Affaire)
      - Passager : Marc
      - Prix : 20000$
      - Trajet : YUL - YYZ
      - Utilisation restantes: 10
    - Billet f3 (Classe : Affaire)
      - Passager : Marc
      - Prix : 18000$
      - Trajet : YUL - YYZ
      - Utilisation restantes: 10

  - Points : 6060
  - Coupons :

- Membre John:
  - Billets :
    - Billet A1B2C3 (Classe : Premium economie)
      - Passager : John
      - Prix : 1000$
      - Trajet : YUL - YYZ
    - Billet f1 (Classe : Premium economie)
      - Passager : John
```

```
Select Microsoft Visual Studio Debug Console
- Membre John:
  - Billets :
    - Billet A1B2C3 (Classe : Premium economie)
      - Passager : John
      - Prix : 1000$
      - Trajet : YUL - YYZ
    - Billet f1 (Classe : Premium economie)
      - Passager : John
      - Prix : 10000$
      - Trajet : YUL - YYZ
      - Utilisation restantes: 10
  - Points : 1200
  - Coupons :
- Membre Robert:
  - Billets :
  - Points : 0
  - Coupons :
- Membre Alex:
  - Billets :
    - Billet f1 (Classe : Premium economie)
      - Passager : Alex
      - Prix : 10000$
      - Trajet : YUL - YYZ
      - Utilisation restantes: 10
    - Billet A1B2C3 (Classe : Premium economie)
      - Passager : Alex
      - Prix : 895.275$
      - Trajet : YUL - YYZ
  - Points : 299
  - Coupons :
- Points cumulee: 1189
- Jours premium restant: 30
```

Spécifications générales

- Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.
- Ne remettez que les fichiers .h et .cpp lors de votre remise.
- Documenter votre code source.
- **Bien lire le barème de correction ci-dessous.**

Questions

Répondez aux questions au début du main.

1. Pourquoi a-t-on besoin de l'attribut `type_` dans la classe membre? Que ce serait-il passé s'il n'existait pas?
2. Quelle est l'importance de l'utilisation d'un `static_cast` ?

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- **(3 points) Compilation du programme**
- **(3 points) Exécution du programme**
- **(4 points) Comportement exact des méthodes du programme**
- **(3 points) Utilisation adéquate de l'héritage**
- **(2 points) Documentation du code. Qualité du code**
- **(3 points) Utilisation adéquate de `static_cast` (héritage)**
- **(2 points) Réponse aux questions.**