**INF1010**

**Programmation Orientée-Objet**

**Automne 2019**

# Travail pratique #4
# Polymorphisme, Héritage multiple

---

**Objectifs :** Permettre à l'étudiant de se familiariser avec le polymorphisme, la conversion dynamique de type, l'héritage multiple, les fonctions virtuelles, les classes abstraites et les interfaces.

**Remise du travail :** Mardi 22 octobre 2019, 8h (AM)

**Références :** Notes de cours sur Moodle & Chapitre 19 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

## Introduction

Le travail consiste à continuer l'application pour le programme de fidélité d'Air Poly commencée aux TPs précédents en y intégrant intégrant le polymorphisme, la conversion dynamique de type, l'héritage multiple, les fonctions virtuelles et les classes abstraites.

Les fonctions virtuelles permettent à une classe dérivée de surcharger une méthode et que celle-ci soit appelée par le compilateur C++ sur un pointeur de cette classe, même si le pointeur a été défini comme un pointeur de la classe mère. De plus, elles peuvent servir à définir des classes abstraites en terminant la définition d'une fonction virtuelle par un = 0. Une classe ayant au moins une méthode abstraite est une classe abstraite et on ne peut pas créer d'objet de cette classe.
Une classe possédant seulement des fonctions virtuelles pures ne peut être instanciée et est appelée une interface.

Afin d'attirer plus de clients, Air Poly décide de faire des soldes sur la vente des billets. On introduit donc les classes BilletRegulierSolde, FlightPassSolde et Solde. Air Poly sollicite votre aide pour introduire ce nouveau système de soldes.

Pour vous aider, les fichiers du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes ou les méthodes à modifier décrites plus bas. Les attributs ou méthodes qui ne sont plus nécessaires ont été supprimés.

## Spécifications générales

- Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.
- Vous devez tirer parti du polymorphisme. **Tout usage de static_cast, typeof ou un attribut spécifiant le type d'une classe sera pénalisé**. Vous devez aussi limiter la duplication de code en utilisant les méthodes des classes mères lorsque possible.
- Utiliser la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.
- L'affichage du programme doit être identique à celui présenté en annexe.
- Documentez votre code source.

## Modifications apportées à l'application

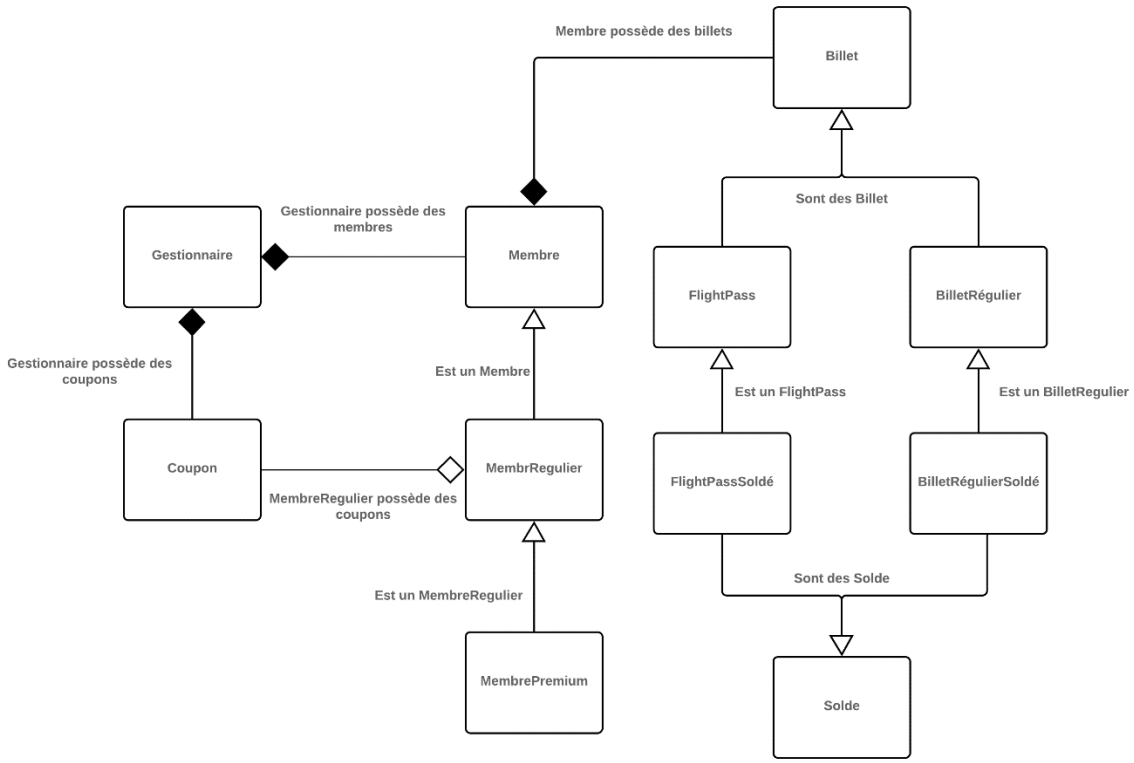
Quelques modifications ont été apportées à l'application pour permettre de simplifier le code.

- Les constructeurs de la classe Billet et de ses dérivées ne prennent plus le nom du passager en paramètre. C'est maintenant la méthode Membre::ajouterBillet() qui se charge d'assigner le nom du détenteur du billet.
- Plutôt que passer tous les attributs des billets, membres ou coupons en paramètre dans les méthodes Gestionnaire::ajouterMembre(), ajouterCoupon(), assignerBillet() et la méthode acheterCoupon() de la classe Billet et toutes ses dérivées, on passe maintenant un pointeur vers ces objets. Les billets, coupons et membres sont donc créés dans le main et non dans

Gestionnaire ou Membre. Ceci ne change rien aux relations de composition et d'agrégation entre les classes.

## Aperçu des classes

Le diagramme de classe ci-dessous peut vous aider à visualiser les interactions entre les classes. Un losange noir représente une relation de composition entre deux classes. Un losange blanc représente une relation d'agrégation entre deux classes. Une flèche pointue représente une relation d'héritage entre deux classes.




## Travail à réaliser

Veuillez suivre les instructions indiquées pour chaque classe. **Vous devez décider si une méthode doit être virtuelle ou virtuelle pure**. Aucune indication n'est donnée dans les instructions indiquant si c'est à faire. Si vous remarquez qu'une même méthode se répète dans plusieurs classes ayant une relation d'héritage, demandez-vous si elle devrait être virtuelle.

## Classe Solde

Cette classe représente un objet en solde. Elle doit être **abstraite**. Elle possède un attribut qui indique le pourcentage de solde à être appliqué sur un objet.

**La méthode suivante doit être modifiée :**

- getPrixBase() : Cette méthode n'a aucune implémentation dans la classe Solde. Ce sont les classes qui dérivent de Solde qui vont donner une implémentation à cette méthode.

### Classe Billet

Cette classe est maintenant **abstraite**. Elle ne peut plus être instanciée. De plus, vous remarquerez que les constructeurs de Billet ne prennent plus le nom du passager en paramètre. C'est la méthode Membre::ajouterBillet() qui s'occupe d'assigner le nom du passager à un billet. L'attribut typeBillet_ a été retiré.

**Les méthodes suivantes doivent être implémentées/modifiées :**

- clone() : Cette méthode n'a aucune implémentation dans la classe Billet. Ce sont les classes qui dérivent de Billet qui vont donner une implémentation à cette méthode.
- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

### Classe BilletRegulier

**Les méthodes suivantes doivent être implémentées :**

- clone() : Cette méthode retourne un objet alloué dynamiquement qui est une copie de l'objet courant.
- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

### Classe BilletRegulierSolde

Cette classe représente un billet régulier qui est en solde. L'attribut prix_ de la classe Billet est le prix original du billet, mais la classe Solde apporte l'attribut pourcentageSolde_ qui est le pourcentage de solde sur le billet.

**Les méthodes suivantes doivent être implémentées :**

- Le constructeur par paramètres.
- getPrix() : Cette méthode retourne le prix du billet. Puisque le billet est en solde, vous devez retourner le prix en tenant compte du solde.
- getPrixBase() : Cette méthode retourne le prix de base du billet avant le solde.
- clone() : Cette méthode retourne un objet alloué dynamiquement qui est une copie de l'objet courant.
- afficher() : Cette méthode permet d'afficher les informations du billet. L'affichage doit correspondre à celui donné en annexe.

## Classe FlightPass

**Les méthodes suivantes doivent être implémentées :**

- clone() : Cette méthode retourne un objet alloué dynamiquement qui est une copie de l'objet courant.
- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

## Classe FlightPassSolde

Cette classe représente un flight pass qui est en solde. L'attribut prix_ de la classe Billet est le prix original du flight pass, mais la classe Solde apporte l'attribut pourcentageSolde_ qui est le pourcentage de solde sur le flight pass.

**Les méthodes suivantes doivent être implémentées :**

- Le constructeur par paramètres.
- getPrix() : Cette méthode retourne le prix du flight pass. Puisque le flight pass est en solde, vous devez retourner le prix en tenant compte du solde.
- getPrixBase() : Cette méthode retourne le prix de base du billet avant le solde.
- clone() : Cette méthode retourne un objet alloué dynamiquement qui est une copie de l'objet courant.
- afficher() : Cette méthode permet d'afficher les informations du flight pass. L'affichage doit correspondre à celui donné en annexe.

## Classe Membre

L'attribut typeMembre_ a été retiré.

**La méthode suivante doit être implémentée :**

- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

**Les méthodes suivantes doivent être modifiées :**

- Le constructeur par copie. Ce constructeur utilisait l'attribut typeMembre_ et static_cast pour faire des copies des billets. Vous devez adapter ce constructeur en utilisant une autre technique pour faire des copies des billets.
- utiliserBillet() : Cette méthode utilisait l'attribut typeBillet_ et static_cast. Vous devez adapter cette méthode pour éliminer l'usage de static_cast.
- operator= : Cet opérateur utilisait l'attribut typeBillet_ et static_cast. Vous devez adapter cet opérateur en utilisant une autre technique pour faire des copies des billets.

## Classe MembreRegulier

Lorsqu'un membre achète un billet en solde, on calcule le nombre de points que celui-ci rapporte avec le solde appliqué sur le prix de base du billet. Ceci ne nécessite aucune modification de la méthode calculerPoints(), mais un mauvais usage du polymorphisme entrainera un mauvais résultat.

**Les méthodes suivantes doivent être implémentées :**

- peutAcheterCoupon() : Cette méthode retourne vrai si le nombre de points du membre est supérieur ou égal au coût du coupon en paramètre. Elle retourne faux autrement.
- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

## Classe MembrePremium

La méthode calculerCoutCoupon() a été implémentée pour vous et retourne le nombre de points nécessaire pour acheter un coupon en tenant compte qu'un membre premium a un rabais sur l'achat de coupons.

**Les méthodes suivantes doivent être implémentées :**

- peutAcheterCoupon() : Cette méthode retourne vrai si le membre a assez de points pour acheter le coupon en paramètre. Elle retourne faux autrement. Vérifiez si le nombre de points du membre est supérieur ou égal au résultat de l'appel de la méthode calculerCoutCoupon().
- afficher() : Cette méthode remplace l'opérateur de sortie. Fiez-vous sur le code existant de l'opérateur de sortie pour votre implémentation. L'affichage doit correspondre à celui donné en annexe.

## Classe Gestionnaire

Puisque les billets ne sont plus alloués dans la méthode assignerBillet(), vous remarquerez que cette méthode modifie le prix du billet si jamais on applique un coupon ou le rabais du membre premium.

**Les méthodes suivantes doivent être implémentées :**

- calculerRevenu() : Cette méthode doit parcourir tous les membres et tous les billets. Elle fait la somme du prix de chaque billet. N'oubliez pas que certains produits peuvent être soldés. Si un produit est soldé, on veut son prix en tenant compte du solde. Si vous utilisez bien le polymorphisme, cette méthode devrait être très simple car la méthode getPrix() des classes dérivées de Solde fait ce traitement.
- calculerNombreBilletsEnSolde() : Cette méthode doit parcourir tous les membres. Elle retourne le nombre total de billets que les membres possèdent qui sont en solde.
- afficher() : Cette méthode permet d'afficher les informations du gestionnaire. L'affichage doit correspondre à celui donné en annexe.

**Les méthodes suivantes doivent être modifiées :**

- assignerBillet() : Cette méthode utilise présentement l'attribut typeMembre_ et static_cast. Vous devez l'adaptez pour ne plus utiliser static_cast et cet attribut non-existant. De plus, vous devez bien traiter le cas où un billet est en solde. S'il est en solde, on veut appliquer le coupon et possiblement le rabais des membres premiums **sur le prix de base.** Il ne faut pas calculer les rabais sur le prix après solde. Puisque cette méthode modifie le prix du billet lorsqu'il y a un coupon ou un rabais de membre premium, vous devez vous assurer de modifier le prix de base du billet sans tenir compte du rabais.

## Fichier 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. Vous pouvez utiliser ce fichier pour mieux comprendre les requis des fonctions.

## Affichage

Votre affichage doit correspondre à celui ci-dessous.

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

- Membre Marc:
        - 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
                - 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
                        - Vol le   : 2019-12-21
                - Billet f1 (Classe : Premium economie)
                        - Passager : John
                        - Prix     : 10000$
                        - Trajet   : YUL - YYZ
                        - Utilisation restantes: 10
                - Billet G7U8E2 (Classe : Economie)
                        - Passager : John
                        - Prix     : 2250$
                        - Trajet   : YUL - YYZ
                        - Vol le   : 2019-12-21
                        - Pourcentage solde: 0.5%
        - Points : 412
        - Coupons :
- Membre Robert:
```

```
- Membre Robert:
        - Billets :
                - Billet T8U7P0 (Classe : Economie)
                        - Passager : Robert
                        - Prix     : 4500$
                        - Trajet   : YUL - YYZ
                        - Vol le   : 2019-12-21
                        - Pourcentage solde: 0.25%
        - Points : 337
        - 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
                        - Vol le   : 2019-12-21
                - Billet T5R2F4 (Classe : Economie)
                        - Passager : Alex
                        - Prix     : 9940.55$
                        - Trajet   : YUL - YYZ
                        - Utilisation restantes: 10
                        - Pourcentage solde: 0.1%
                - Billet J9K5L0 (Classe : Economie)
                        - Passager : Alex
                        - Prix     : 22186.9$
                        - Trajet   : YUL - YYZ
                        - Utilisation restantes: 10
                        - Pourcentage solde: 0.5%
        - Points : 2128
        - Coupons :
        - Points cumulee: 3892
        - Jours premium restant: 30
```

## Questions

Répondez aux questions au début du main.

1. Nous remarquons que le classe Solde à une méthode sans implémentation, getPrixBase(). Dans le contexte du polymorphisme, quelle est l'utilité d'avoir une méthode dans une classe parent qui ne possède aucune implémentation?
2. Pour quelle raison devons-nous déclarer certains destructeurs virtuels?

## Correction

La correction du TP4 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme
- (3 points) Exécution du programme
- (3 points) Comportement exact des méthodes du programme
- (3 points) Utilisation du polymorphisme et de dynamic_cast
- (2 points) Documentation du code et bonne norme de codage
- (2 points) Utilisation correcte du mot-clé *this* et *const*
- (2 points) Gestion adéquate de la mémoire (destructeurs, copies, etc.)
- (2 points) Réponses aux questions