INF1010

Programmation Orientée-Objet

Travail pratique #4
Fonctions virtuelles et polymorphisme

Objectifs :	Permettre à l'étudiant de se familiariser avec les fonctions virtuelles et le concept de polymorphisme.
Remise du travail :	Lundi le 5 novembre 2018, 8h
Références :	Notes de cours sur Moodle & Chapitre 19 du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que les fichiers .cpp et .h complétés 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 parti du même groupe. Veuillez suivre le guide de codage

Le travail effectué dans ce TP continue celui amorcé par les TP1, 2 et 3, en y intégrant les notions de fonctions virtuelles et de polymorphisme.

Les fonctions virtuelles permettent à une classe fille 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 fonctions virtuelles pures en terminant la définition d'une fonction virtuelle par un = 0. Une classe possédant au moins une fonction virtuelle pure ne peut être instanciée et est appelée une interface.

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

ATTENTION : Plusieurs modifications ont été apportés par rapport au TP3, veuillez lire l'énoncé attentivement.

- Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.
- Vous devez tirer parti du polymorphisme. L'utilisation de solutions alternatives (attribut type_, typeof, static_cast, etc.) sera jugée hors-sujet et sanctionnée.
- Vous devez ajouter des destructeurs, surcharger l'opérateur = et ajouter un constructeur par copie chaque fois que cela vous semble pertinent. N'en ajoutez pas si cela ne vous semble pas pertinent.
- Il faut utiliser le mot clé const chaque fois que cela vous semble pertinent.
- L'affichage du programme doit être aussi proche que possible que celui présenté en annexe.
- Il faut ajouter des commentaires aux méthodes jugées complexes
- Vous ne devez pas modifier les signatures des méthodes fournies.

Dépense

La classe ne contient plus de classes dérivées. L'ancienne classe DepenseIndividuelle n'existe plus, et Depense a pris la place de DepenseGroupe.

L'attribut type_ ainsi que l'énum correspondant n'existent plus.

Utilisateur

L'attribut intérêt a été remplacé par les concepts de balance, qui sont explicités dans la description des classes ci-dessous.

Elle contient les nouveaux attributs courriel_ et idPaypal_ qui sont utilisés pour les transferts.

UtilisateurPremium ne contient plus l'attribut taux_, qui est maintenant géré par les transferts.

Transfert

Possède 2 nouveaux enfants TransfertPaypal et TransfertInterac. Elle possède aussi la responsabilité de réaliser des transferts d'argent, et c'est dans celle-ci que se fait le calcul des frais de transfert. Elle met aussi à jour les balances des utilisateurs.

Groupe

Les dépenses sont maintenant en composition avec groupe au lieu d'être en agrégation. C'est maintenant dans le groupe que doivent être créées les dépenses.

La méthode ajouterDepense ne prend plus de vecteur payePour. Une dépense est donc ajoutée à tous les membres du groupe.

Main

Le fichier main contient maintenant des tests qui vous sont fournis afin de valider le fonctionnement de votre programme. Les tests ne devraient pas être modifiés et seront utilisés lors de la correction.

Général

Il ne vous sera pas indiqué s'il faut ou non utiliser des méthodes virtuelles ou virtuelles pures. C'est à vous de décider.

Vous pouvez utiliser les tests du main.cpp pour comprendre les particularités des méthodes.

Comme précisé dans les directives du TP, vous devez implémenter les destructeurs, les constructeurs par copie et les opérateurs = seulement quand cela est pertinent.

Depense

- Implémenter le constructeur

Utilisateur, UtilisateurPremium et UtilisateurRegulier

Un utilisateur est soit premium, soit régulier.

Un utilisateur premium perçoit un retour de 3% sur chacun de ses transferts et ne paie pas de frais de transfert.

Il contient les nouveaux attributs suivants :

- balanceTransferts : représente le montant que l'utilisateur doit payer ou recevoir dans les transferts suivant un équilibrage des comptes. Il peut être positif s'il doit transférer de l'argent aux autres utilisateurs du groupe, ou négatif s'il doit en recevoir de la part d'autres utilisateurs.
- balanceFrais : représente le montant des frais que l'utilisateur doit payer ou recevoir suite aux transferts réalisés par un équilibrage des comptes. Il peut être positif s'il doit payer des frais (utilisateur régulier) ou négatif s'il reçoit des retours sur ses transferts (utilisateur premium).

Les méthodes suivantes doivent être implémentées:

- Les méthodes *print* doivent être implémenter de sorte à afficher les informations de l'usager ainsi que ses dépenses. L'affichage doit correspondre au modèle donné. Vous devez tirer parti de l'héritage.

Transfert

Représente un transfert, qui est soit un transfert Interac, soit un transfert PayPal, dépendamment du profil de l'utilisateur qui réalise le transfert.

Les méthodes suivantes doivent être implémentées :

- getFraisTransfert : calcule les frais associés au transfert. Les frais dépendent de la nature du transfert. Il faut tirer parti de l'héritage pour le calcul.
- effectuerTransfert : effectue un transfert d'argent entre l'expéditeur du transfert et le receveur. Il ajoute donc de l'argent dans la balance des transferts de l'utilisateur expéditeur, et diminue la balance des transferts de l'utilisateur receveur. Il modifie aussi les frais à payer ou les retours à recevoir de l'expéditeur.

TransfertInterac

Cette classe possède l'attribut courriel, qui correspond au courriel du receveur du transfert

- getFraisTransfert : calcule les frais associés au transfert, soit un montant fixe de 1$ par transfert pour les utilisateurs régulier.

TransfertPaypal

Cette classe possède l'attribut id, qui correspond au courriel de l'expéditeur du transfert

- getFraisTransfert : calcule les frais associés au transfert, soit 2.26% du montant plus des frais fixe de 0.30$ par transfert pour les utilisateurs régulier.

Groupe

Les méthodes suivantes doivent être implémentées :

- L'opérateur+= prenant un Utilisateur* et l'ajoute à la liste des utilisateurs du groupe. Il faut vérifier que les utilisateurs peuvent bien être ajoutés (validité de l'abonnement et appartenance à un groupe)
- La méthode ajouterDepense : prend en paramètre l'Utilisateur* ayant réalisé la dépense et les informations de la dépense. Cette méthode doit vérifier que l'utilisateur appartient bien au groupe, créer une nouvelle dépense et l'ajouter au groupe, puis mettre à jour la valeur des comptes de tous les utilisateurs du groupe.
- La méthode equilibrerComptes doit être complétée afin de créer les bons types de transfert et les effectuer.

Main

Le fichier main.cpp vous est fourni avec les tests. Modifiez-le seulement s’il y a de la mémoire à libérer.

Considérez le code suivant :

```
class A {
public:
     void f() const { cout << 1; }
};

class B : public A {
public:
     virtual void f() const { cout << 2; }
};

class C : public B {
public:
     void f() const { cout << 3; }
};

int main()
{
    vector<A*> v;
    B b;
    v.push_back(&b);
    v[0]->f();
    return 0;
}
```

1. Quelle sera la sortie du code ci-dessus?
2. Pourquoi?
3. Si le vecteur v était de type vector<B*> et la variable b était de type C, proposez deux solutions pour que la sortie soit celle de la méthode B::f().
4. Dans ce TP, pourquoi ne peut-on pas instancier un objet de type Transfert?

La correction du TP se fera sur 20 points.

- (3 points) Compilation
- (3 points) Exécution
- (3 points) Comportement
- (3 points) Utilisation du polymorphisme et de dynamic_cast
- (2 points) Documentation des méthodes complexes
- (2 points) Utilisation de *const* / du mot clef *this*
- (2 points) Gestion de la mémoire (destructeurs, copies, etc.)
- (2 points) Réponse aux questions.

```
Erreur : L'utilisateur Rebecca doit renouveler son abonnement premium
Erreur : L'utilisateur Axel n'est pas un utilisateur premium et est deja dans un groupe.
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!
        Test 27... OK!

Groupe Madrid.
Cout total: 2040$
Utilisateurs:

        - Utilisateur (premium) Ryan :
                Total a payer: -20.00$ (-0.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d1. Prix : 180.00$
                        - Depense (a Montreal) : d9. Prix : 180.00$

        - Utilisateur (premium) David :
                Total a payer: 100.00$ (3.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d3. Prix : 240.00$

        - Utilisateur (premium) Gaspard :
                Total a payer: -20.00$ (-0.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d2. Prix : 360.00$

        - Utilisateur (regulier, dans un groupe) Yves :
                Total a payer:280.00 (7.58$ de frais)
                Depenses :
                        - Depense (a Montreal) : d4. Prix : 60.00$

        - Utilisateur (regulier, dans un groupe) Martine :
                Total a payer:-380.00 (0.00$ de frais)
                Depenses :
                        - Depense (a Montreal) : d5. Prix : 600.00$
                        - Depense (a Montreal) : d8. Prix : 120.00$

        - Utilisateur (regulier, dans un groupe) Samuel :
                Total a payer:40.00 (2.00$ de frais)
                Depenses :
                        - Depense (a Montreal) : d6. Prix : 300.00$


Transferts :
        - Yves     -> Martine  : 280.00$
        - David    -> Martine  : 100.00$
        - Samuel   -> Ryan     : 20.00$
        - Samuel   -> Gaspard  : 20.00$

Press any key to continue . . .
```