

INF1010

Programmation Orientée-Objet

Travail pratique #4

Fonction virtuelle, polymorphisme et héritage multiple

Objectifs :	Permettre à l'étudiant de se familiariser avec les fonctions virtuelles et le concept de polymorphisme.
Remise du travail :	Lundi 12 mars 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 :	<p>Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.</p> <p>Veuillez suivre le guide de codage</p>

Introduction

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

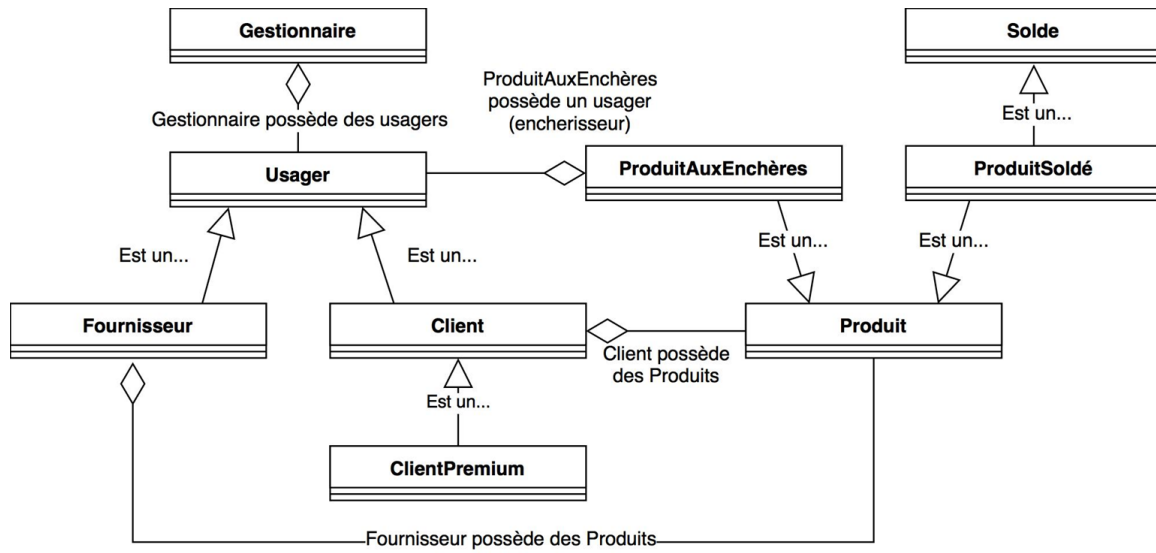
Les fonctions virtuelles permettent à une classe fille de surcharger une méthode et que celle-ci soit appelée par le compilateur C++ sur un pointeur de cette classe, même si le pointeur a été défini comme un pointeur de la classe mère. De plus, elles peuvent servir à définir des fonctions virtuelles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

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.

Directives

- 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 `const` chaque fois que cela vous semble pertinent.
- L'affichage du programme doit être identique à celui présenté en annexe.
- Ce TP fait intervenir des méthodes qui nécessitent d'être commentées. L'absence de commentaires dans une méthode jugée « complexe » sera sanctionnée.

Aperçu du projet - Hiérarchie des classes



Description des classes

La classe Gestionnaire contient les différents usagers et permet d'effectuer des actions groupées sur ceux-ci. C'est également cette classe qui est en charge de gérer les enchères.

La classe Usager représente un utilisateur. Cette classe devrait être abstraite.

La classe Client représente un usager capable d'acheter des produits.

La classe ClientPremium représente un client bénéficiant d'un statut particulier. **Les clients premium bénéficient notamment d'une réduction de \$5 sur tous leurs achats.** Si un item coûte moins que \$5, le prix est alors de zéro pour un client premium (il ne peut pas être négatif).

La classe Fournisseur représente un Usager qui propose des produits à la vente. Il possède un catalogue de produits qu'il met en vente. Un fournisseur ne peut pas acheter de produits.

La classe Produit représente un produit ordinaire (la classe Produit Ordinaire est fusionnée avec la classe de base Produit) . Cette classe n'est pas abstraite. Un produit ordinaire peut être immédiatement ajouté dans le panier d'un client. Un produit connaît l'adresse mémoire de son fournisseur.

La classe ProduitSolde représente un produit ordinaire sur lequel s'applique un rabais. **Pour les clients premiums, le rabais s'applique avant la réduction de \$5.**

La classe Solde, caractérisée par l'attribut *pourcentageRabais_*, représente un rabais. La classe ProduitSolde hérite de la classe Solde et de la classe Produit..

La classe ProduitAuxEncheres représente un produit étant mis aux enchères. Pour qu'un client mette un tel produit dans son panier, il doit surenchérir. Un produit aux enchères voit son prix évoluer au fil des enchères. Il ne peut être que dans un seul panier à la fois. On a ajouté l'attribut qui permet de stocker l'adresse mémoire du client de la dernière mise.

Remarque: La classe Panier a été retirée et n'existe plus dans ce TP. Il en découle que:

1. Cette classe faisait office de couche intermédiaire entre Client et Produit. Ses responsabilités ont donc été transférées à Client, qui possède désormais un attribut *panier_* (vecteur de produits).
2. La classe Panier était intéressante pédagogiquement, car elle vous a permis de pratiquer la composition par pointeur. Dans ce TP, il n'y a désormais plus aucun cas de composition par pointeur. Pensez donc à revoir vos premiers TP lorsque vous vous préparerez au final.


Travail à réaliser

Général

Il ne vous sera pas indiqué lorsqu'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.

Gestionnaire

- ~~afficherLesProfils~~ doit afficher le profil de tous les utilisateurs enregistrés dans le gestionnaire. L'affichage doit correspondre au modèle donné.
- ~~obtenirChiffreAffaires~~ doit retourner la somme du total à payer de tous les usagers enregistrés dans le gestionnaire. Un fournisseur n'a rien à payer (total à payer = 0).
- ~~ajouterUsager~~ doit ajouter un usager au gestionnaire (attribut usagers) si celui-ci n'y figure pas encore.
- ~~reinitialiser~~ doit vider les paniers des clients et vider les catalogues des fournisseurs. Lorsqu'on enlève un produit d'un catalogue, il faut remettre à *nullptr* l'attribut *fournisseur_* du produit. 
- ~~encherir~~ doit se charger de surenchérir. Le premier paramètre correspond au client qui surenchérit, le second paramètre correspond au produit sur lequel le client surenchérit, et le troisième paramètre correspond au prix que le client est prêt à mettre. Si le prix de surenchère est inférieur ou égal au prix précédent, il ne se passe rien. Sinon, on utilise la fonction *ProduitAuxEncheres::mettreAJourEnchere*.

Usager, Client, ClientPremium, Fournisseur

- Les méthodes ~~afficherProfil~~ doivent être implémentées de sorte à afficher des informations précises sur l'usager. L'affichage doit correspondre au modèle donné. Il faut tirer parti de l'héritage.
- Les méthodes ~~obtenirTotalAPayer~~ doivent être implémentées de sorte à ce qu'elles retournent le montant total du panier de l'usager. Pour les clients premiums, le rabais de \$5 sur chaque article doit être pris en compte. Il faut tirer parti de l'héritage.
- Les méthodes ~~reinitialiser~~ doivent être implémentées de sorte à ce qu'elles réinitialisent le catalogue/panier de l'usager. Les méthodes doivent vider le panier

~~si c'est un client et vider le catalogue si c'est un fournisseur. Lorsqu'on enlève un produit d'un catalogue, il faut remettre à *nullptr* l'attribut *fournisseur_* du produit.~~

Lorsqu'on vide le panier d'un client et que le panier contenait des produits mis aux enchères, il faut mettre à *nullptr* l'attribut *encherisseur_* de ceux-ci et il faut remettre le *prix_* au *prixInitial_*. **Pour faire cela, on vous impose d'utiliser la fonction *dynamic_cast*.**

Produit, ProduitSolde, ProduitAuxEncheres

- Le produit n' a plus l' attribut ayant une référence sur un Fournisseur, mais plutôt un pointeur contenant l'adresse mémoire du Fournisseur permettant ainsi de changer de le fournisseur du produit.
- Les méthodes *obtenirPrix* doivent être implémentées de sorte à ce qu'elles renvoient le prix du produit. Si le produit est soldé, il faut appliquer le rabais de la classe Solde. Si le produit est mis aux enchères, il faut retourner le prix de la dernière offre (attribut *prix_*).
- Les méthodes *afficher* doivent afficher les informations sur le produit selon le format donné.

ProduitAuxEncheres

- La méthode *mettreAJourEnchere* doit être implémentée. Le premier paramètre correspond au client qui souhaite surenchérir et le second paramètre correspond au prix qu'il est prêt à investir.

La méthode doit d'abord vérifier que le client n'est pas en train de surenchérir sur sa propre offre. Si c'est le cas, il ne se passe rien.

La méthode doit ensuite mettre à jour le prix du produit et ajouter le produit au panier du client.

Finalement, la méthode doit enlever le produit du panier de l'enchérisseur précédent puis mettre à jour l'attribut *encherisseur_*.

Questions

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 `Usager`?
5. À partir de la classe de base `Produit`, on dérive les classes `ProduitsAuxEncheres` et `ProduitSolde` qui dérive aussi de la classe de base `Solde`. Donc la classe `ProduitSolde` a un héritage multiple. Il serait intéressant de connaître (afficher par exemple) séparément l'ensemble des `Produits` et l'ensemble des `Produits Soldés`. Expliquer comment peut-on le faire ? Vous pouvez aussi écrire le code.

Correction

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.

Annexe : Affichage attendu

```
PANIER (de Donada--Vidal)
  chaussures
    reference: 1
    prix: $40
  montre
    reference: 4
    prix: $100
    rabais: 30%

PANIER (de Cash)
  chaussures
    reference: 1
    prix: $40
  montre
    reference: 4
    prix: $100
    rabais: 30%
  violoncelle
    reference: 3
    prix: $21000
    prix initial: 5000
    enchereur: Cash
  nem crevettes
    reference: 5
    prix: $2
    rabais: 0%

PANIER (de Doe)

CATALOGUE (de Bellaiche)

CATALOGUE (de Kadoury)
  chaussures
    reference: 1
    prix: $40
  montre
    reference: 4
    prix: $100
    rabais: 30%

CATALOGUE (de Doe)
```


PROFILS

Donada--Vidal, Gaspard (5215487)
code postal: P4R 1I5
code client: 1997
panier: 2 elements

S, Rick (8435174)
code postal: HF1 8H3
code client: 20012003
panier: 0 elements

Bellaiche, Martine (6845123)
code postal: H4C 8D4
catalogue: 0 elements

Kadoury, Samuel (1687421)
code postal: H1G 2G4
catalogue: 2 elements

Doe, John (0)
code postal: A1A A1A
code client: 0
panier: 0 elements

Doe, John (0)
code postal: A1A A1A
code client: 0
panier: 0 elements
jours restants: 0

Doe, John (0)
code postal: A1A A1A
catalogue: 0 elements

Cash, Julie (1126250)
code postal: HZ9 1J4
code client: 19141918
panier: 4 elements
jours restants: 50

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!  
Test 28... OK!  
Test 29... OK!  
Test 30... OK!  
Test 31... OK!  
Test 32... OK!  
Test 33... OK!  
Test 34... OK!  
Test 35... OK!  
Test 36... OK!  
Test 37... OK!  
Test 38... OK!  
Test 39... OK!  
Test 40... OK!  
Test 41... OK!  
Test 42... OK!  
Test 43... OK!  
Test 44... OK!
```