INF1010

Programmation Orientée-Objet

Travail pratique #3
Héritage

Objectifs :	Permettre à l'étudiant de se familiariser avec l'héritage.
Remise du travail :	Lundi le 26 fevrier 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 partie du même groupe. Veuillez suivre le guide de codage

Le travail consiste à continuer l'application de vente en ligne commencé au TP1 et 2 et d'y intégrer les notions d'héritage. On notera que la classe Rayon a été supprimée pour ce TP.

L'héritage permet de créer une classe de base possédant ses propres caractéristiques et de l'utiliser pour créer de nouvelles classes auxquelles on peut attribuer la caractéristique «est un ».

Dans ce TP, vous serez amené à transformer le code du TP2 pour prendre en considération deux hiérarchies d'héritage :

- Produit : produit vendu directement au client et produit vendu aux enchères.
- Usager : usager client et usager fournisseur.

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 objet d'une classe en objet d'une autre classe. Dans le cas du TP, par exemple il servira à transformer un pointeur du type Produit vers un pointeur du type réel de ce produit (ProduitOrdinaire, ProduitAuxEncheres).

Pour vous aider, les fichiers du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes et les modifications (attributs et méthodes) décrites plus bas.

ATTENTION : 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.

ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP2.

Classe *Produit*

La classe de base `Produit` caractérise un produit quelconque. On part de l'hypothèse qu'un produit a un prix unique et un seul fournisseur. Dès que l'on crée un objet Produit, il connaît automatiquement le fournisseur qui possède cet objet.

Cette classe possède en plus les attributs suivants :

`Fournisseur & fournisseur_` : Attribut permettant d'identifier le fournisseur du produit.
`TypeProduit type_;` attribut lors de la création d'un objet de la classe dérivée permet de connaître le type des objets dérivés.

Les méthodes
`Le constructeur` : il faut modifier le constructeur pour initialiser le fournisseur et le type, ainsi qu'ajouter le produit au fournisseur.
`Fournisseur & obtenirFournisseur()` : retourne le fournisseur associé au produit.
`TypeProduit retournerType()` : retourne la valeur de l'attribut `type_`.
`~Produit()` : doit retirer le produit du catalogue du Fournisseur.

Classe *ProduitOrdinaire*

La classe `ProduitOrdinaire` caractérise un produit qui peut être acheté directement par un client. Cette classe hérite de la classe `Produit`.

Cette classe contient l'attribut privé suivant :

- `estTaxable_` : un booléen qui indique si le produit est taxable ou non.

Les méthodes:

- ~~Un constructeur par défaut et par paramètres qui utilise le constructeur par paramètres de Produit et initialise également l'attribut `estTaxable_` par la valeur donnée en paramètre et par défaut (true).~~
- ~~La méthode d'accès et de modification de l'attribut `estTaxable_` .~~
- La surcharge de l'opérateur >> qui fait appel à la surcharge de l'opérateur >> de la classe Produit et ajoute la lecture de l'attribut `estTaxable_`
- La surcharge de l'opérateur << pour afficher un produit ordinaire. Utilisez une forme d'affichage similaire à la classe Produit. Considérez l'attribut `estTaxable_` dans l'affichage.

Note : Penser à l'utiliser `static_cast`.

Classe *ProduitAuxEncheres*

Cette classe caractérise un produit acheté aux enchères. Cette classe hérite de la classe `Produit`.

Cette classe contient les attributs privés suivants :

- `prixBase_` : le prix misé d'un produit aux enchères.
- `identifiantClient_` : identifiant du client qui a fait la plus grande mise.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres et par défaut utilisant le constructeur de la classe `Produit` et initialisant `le prixBase_` (0 par défaut) et `identifiantClient_` (défaut à 0);
- Les méthodes de modification d'accès aux attributs définis dans cette classe.

La surcharge de l'opérateur >> qui fait seulement appel à la surcharge de l'opérateur >> de la classe `Produit`.

La surcharge de l'opérateur << pour afficher un produit aux enchères. Utilisez une forme d'affichage similaire à la classe `Produit`. Considérez les attributs `prixBase_`et `identifiantClient_` dans l'affichage.

Classe *Usager*

Cette nouvelle classe de base va servir à dériver les classes `Client` et `Fournisseur`. On suppose qu'un usager du site de vente en ligne peut agir comme Client ou comme un Fournisseur. La définition et l'implémentation de cette classe vous sont fournis.

Classe *Fournisseur*

Cette classe caractérise un fournisseur de produits. Elle dérive de la classe `Usager`.

Cette classe contient les attributs suivants :

- `Satisfaction satisfaction_` : enregistrement qui contient un tableau qui conserve la satisfaction d'un client lors d'un achat d'un produit appartenant au fournisseur.
- `vector<Produit*> contenuCatalogue_` : la liste des produits vendus par le fournisseur. Cela correspond à son catalogue de produits.

Les méthodes suivantes doivent être implémentées :

- Le constructeur de fournisseur initialise les attributs de la classe Usager, et tout le tableau de `satisfaction_` a la valeur 0.

- Les méthodes d'accès aux attributs.
- La méthode de modification de l'attribut `satisfaction_`
- La méthode `noter(int appreciation)` : le paramètre `appreciation` représentant l'indice du tableau de l'attribut `satisfaction_` , le tableau est incrémenté à cet indice.
- Les méthodes `ajouterProduit` et `enleverProduit` permettent d'ajouter et retirer un produit du catalogue.
- La surcharge de l'opérateur =. Utiliser la surcharge de l'opérateur = de la classe Usager.

La fonction globale operator <<. Penser à l'utiliser l'opérateur << de la classe de base. Afficher la satisfaction du fournisseur.

Classe *Client*

La classe Client dérive de la classe de base Usager.

Les attributs sont :

```
long dateNaissance_;
Panier *  monPanier_;
```

Les méthodes :

- ~~Le constructeur de Client initialise les attributs de la classe Usager, la date de naissance et le panier à null.~~
- ~~Le destructeur est donné.~~
- ~~Le constructeur de copie est donné.~~
- ~~La surcharge de l'opérateur = est donné~~
- ~~Les méthodes d'accès aux attributs de Client sont données.~~
- ~~La méthode de modification de la date de naissance est donnée.~~
- La méthode `acheter()` doit ajouter un produit ordinaire dans le panier. Si le panier n'existe pas, on doit le créer avec l'identifiant du client (attention, c'est un changement par rapport au tp2) et on l'ajoute au panier. ~~Vous devez écrire les instructions qui permettent au client de donner un note aléatoire de satisfaction du client au fournisseur~~ et ~~faire la mise à jour de la satisfaction au fournisseur~~.
- ~~La méthode `livrer()` est donnée.~~
- ~~La méthode `miserProduit(ProduitAuxEncheres* produitAuxEncheres, double montantMise )` qui permet de miser `montantMise` sur un produit aux enchères donné en paramètre. Cette méthode vérifie~~
  - ✓ Si le prix misé du client est supérieur du prix actuel de l'enchère du produit, alors on met à jour le prix de l'enchère du produit et on change l'identifiant du client dans le produit.
  - ✓ On ajoute le produit dans le panier.

~~La fonction globale operator << . Penser à l'utiliser l'opérateur << de la classe de base.~~ Afficher le contenu du panier du client.

Classe *Panier*

La classe Panier caractérise un panier d'achat. Il contient l'ensemble des produits ordinaires et des produits aux enchères sur lesquels le client a misé. L'utilisation de la fonction **static_cast<T>(Objet)** est utile pour l'implémentation de cette classe.

L'attribut suivant est ajouté dans la classe :

- `idClient_` pour identifier un client associé à ce panier.

Les méthodes suivantes doivent être implémentés ou changées :

- `Panier(id Client)` remplace le constructeur par défaut. Ce constructeur permet d'associer l'id du client qui crée un panier.
- ~~Les méthodes d'accès et de modification pour des attributs sont données~~.
- ~~La fonction `livrer()` est donnée.~~
- ~~La méthode `ajouter(Produit* prod)` est modifiée. Si le produit est ordinaire, on ajoutera la taxe s'il est taxable par la constante TAUX_TAXE du fichier Panier.h. On ajoutera le produit au panier.~~
- ~~La méthode `trouverProduitPlusCher()` est donnée.~~
- La méthode `calculerTotalAPayer()`.
  - ✓ Dans le panier, la méthode cherche si dans le Panier, il existe des `ProduitsAuxEncheres` et si l'identifiant du client qui a donné la plus grande mise pour ce produit correspond avec l'id du client propriétaire de ce panier alors il faut ajouter le montant de ce produit au `totalAPayer_`.
  - ✓ Cette méthode détermine le montant à payer par un client s'il est encore gagnant dans la mise de certains produits.
- La surcharge de l'opérateur << pour afficher un panier. On souhaite dans la surcharge de cet opérateur de considérer qu'on a deux types de produit et utiliser l'opérateur surchargé << adéquat pour chaque produit.

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées. Votre affichage devrait avoir une

apparence semblable au fichier pdf donné. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable ainsi que de choisir vos propres exemples :

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
• Appliquez un l'affichage présenté à la fin de ce document
• Documenter votre code source
• Note : Lorsqu'il est fait référence aux valeurs par défaut, pour un string cela équivaut à la chaîne vide, et pour un entier au nombre 0

Questions

1. Pourquoi est-il logique de dériver une classe ProduitAuxEncheres et une classe ProduitOrdinaire d'une classe Produit?

2. Dans la surcharge de l'opérateur << dans la classe Panier.
    1. Quelle est l'importance de l'utilisation d'un static_cast.
    2. Quel effet aura-t-on si on ne le considère pas dans l'implémentation.

La correction du TP se fera sur 20 points.

Voici les détails de la correction :

- (4 points) Compilation du programme ;
- (4 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) Réponse aux questions.