# INF1010

## Programmation Orientée-Objet

## Travail pratique #1
## Allocation dynamique, composition et agrégation

---

| | |
|---|---|
| **Objectifs :** | Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation |
| **Remise du travail :** | Lundi 05 Février 2018, 8h |
| **Références :** | Notes de cours sur Moodle & Chapitre 2-7 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<br>Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.<br>Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.<br>Veuillez suivre le guide de codage |

NE PAS OUBLIER L'INITIALISATION!!?!??!?!??!??!

***La directive de précompilation « #ifndef »***

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'entête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entrainerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H
#define NOMCLASSE_H
// Définir la classe NomClasse ici
#endif
```

**La directive de précompilation « #include»**

La directive de précompilation pour l'inclusion de fichiers « #include »

```
1. #include <nom_fichier>
2. #include "nom_fichier"
```

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

## Mise en contexte

En tant que jeune entrepreneur, vous avez eu l'idée de lancer un site internet de vente en ligne spécialisé dans la vente du matériel de toute catégorie, ce site propose aux clients des produits de toutes gammes dans différentes catégories. Dans un premier temps, vous serez amené à remplir

les informations liées à chaque produit et à le classer dans le rayon correspondant. Par la suite. des clients pourront acheter des produits et les stocker dans leur panier d'achat et livrer ce panier.

## Travail à réaliser

Les fichiers Produit.h, Rayon.h, Panier.h et et Client.h vous sont fournis. Vous devez compléter les fichiers .cpp, et le fichier main.cpp en suivant les directives fournies.

.

## Classe *Produit*

Cette classe Produit est caractérisée par un nom, une référence, et un prix.

Cette classe contient les attributs privés suivants :

- Un nom (string).
- Une référence du produit (entier).
- Un prix (double).

Les méthodes suivantes doivent être implémentées :

- ~~Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. Les valeurs par défaut~~
  - ~~le nom « outil ».~~
  - ~~la référence, et prix à 0.~~
- ~~Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.~~
- ~~Les méthodes d'accès aux attributs.~~
- Les méthodes de modification des attributs.✓
- Une méthode afficher() qui affiche l'état des attributs.✓

## Classe Rayon

Cette classe est caractérisée par une catégorie et un tableau dynamique de Produit.

Cette classe contient les attributs privés suivants :

- Le nom de la catégorie (string).
- Un tableau dynamique de pointeurs à un objet Produit (Produits**).
- La capacité du tableau dynamique (entier).
- Le nombre de produits (entier).

Les méthodes suivantes doivent être implémentées dans le fichier .cpp

- Un constructeur par défaut qui initialise l'attribut aux valeurs par défaut : ~~catégorie = « inconnu »~~, le tableau à null, ~~la capacité et le nombre de produits à 0.~~

- ~~Un constructeur par paramètres qui initialise l'attribut catégorie, le tableau à null, la capacité et le nombre de produits à 0.~~
- ~~Les méthodes d'accès aux attributs.~~
- La méthode de modification de l'attribut catégorie. ✓
- La méthode ajouterProduit () qui ajoute le pointeur d'un objet produit au tableau dynamique. Si le tableau est null, on crée le tableau initial avec une capacité de 5. Si le tableau est non null et que la capacité est atteinte, on augmente la capacité de 5 du tableau dynamique.
- La méthode afficher() qui affiche l'état des attributs.

## Classe *Panier*

Cette classe sert à regrouper les produits achetés par un client, à calculer leur nombre de produits et le total à payer.

Cette classe contient les attributs privés suivants :

- Un tableau dynamique de pointeurs à des objets produits (produit).
- Un compteur (entier) qui permettra de connaitre le nombre de produits .
- La capacité de mémoire du tableau dynamique.
- Un total à payer (entier).

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètre qui initialise la capacité du tableau dynamique, alloue l'espace mémoire du tableau dynamique et ~~initialise les autres attributs à 0.~~
- ~~Les méthodes d'accès des attributs.~~
- Aucune méthode de modification des attributs.
- Une méthode ajouter () qui prend en paramètre un pointeur à un objet de Produit et l'ajoute au tableau de produits. Lors de l'ajout, si la capacité du tableau est atteinte, on doublera la capacité, pour pouvoir ajouter le pointeur de l'objet Produit. Il faut aussi faire la mise à jour du total à payer.
- Une méthode livrer() qui supprime le contenu du tableau et ré-initialise l'état des autres attributs.
- Une méthode afficher() qui affiche l'état du panier.

## Classe *Client*

Cette classe Client permet de créer un client qui va acheter des produits et les ajouter dans son panier.

### Cette classe contient les attributs privés suivants :

- Un nom_de_client (string).
- Un prénom (string).
- Un code Postal (string) .
- Une date_de_naissance (entier long : les 4 premiers chiffres : année, 2 chiffres suivants : le mois).
- Un identifiant (entier).
- -Un pointeur vers un objet Panier.

### Les méthodes suivantes doivent être implémentées :

- ~~Un constructeur par paramètres qui initialise les attributs et le pointeur de l'objet Panier à null. On note qu'il n'y a pas de constructeur par défaut.~~
- ~~Les méthodes d'accès aux attributs.~~
- Les méthodes de modification des attributs.
- La méthode acheter() ajoute un produit dans le panier du client. Si le panier n'existe pas, on créé un objet Panier de 4 produits, et on ajoute le produit dans le Panier du client.
- La méthode afficherPanier() affiche le contenu du Panier s'il n'est pas vide.
- La méthode livrer() permet de livrer le contenu du panier, en supprimant le Panier actuel du client.

## Main.cpp

Des instructions vous sont fournies dans le fichier Main.cpp et il vous est demandé de les suivre.

Dans votre affichage, vous avez la liberté totale de choisir le design qui vous semble plus ergonomique et plus agréable ainsi que les exemples de produits, de rayon, de client, …etc, à utiliser.

## Directives

- Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Pour chaque classe, on demande d'écrire un constructeur par défaut et par paramètres, vous pouvez en les réunir ensemble si c'est possible.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.
- Appliquez un affichage « user friendly » (ergonomique et joli) pour le rendu final

- Documenter votre code source

## Questions

1. Quel est le lien (agrégation ou composition) entre la classe Client et la classe Panier ? Justifiez.
2. Quel est le lien (agrégation ou composition) entre un Produit et un Rayon ? Justifiez.

## 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 ;
- (3 points) Comportement exact des méthodes du programme ;
- (1 point) Fusion des constructeurs;
- (2 points) Documentation du code et bonne norme de codage ;
- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés ;
- (2 points) Utilisation adéquate des directives de précompilation ;
- (2 points) Allocation et désallocation appropriées de la mémoire ;
- (2 points) Réponse aux questions ;