Université Paris Cité **Langages à objets avancés**
M1 **C++**

Patron Observateur/Observable

Le patron Observateur/Observable est un patron de conception comportemental permettant de définir une dépendance de un à plusieurs entre objets (de l'observable vers ses observateurs), sans pour autant

- qu'un observable ne connaisse à l'avance le nombre ou le type de ses observateurs abonnés,
- ou bien que les classes implémentant les observables ne dépendent des différentes classes d'observateurs possibles.

Intérêt : quand on veut ajouter des observateurs, voire même programmer une nouvelle classe d'observateurs, cela ne demande pas de modifier le code des classes d'observables.

Concrètement, tout changement dans un observable provoque un appel de méthode dans tous ses observateurs abonnés, leur signalant le changement.

Typiquement, on implémente ce patron à l'aide de deux classes abstraites `Observable` et `Observer`. :

- La classe `Observable` contient une liste (privée) d'observateurs de type `Observer`, des méthodes (publiques) permettant à un observateur de s'abonner et de se désabonner à cet observable (c.-à-d. s'ajouter et se retirer de la liste), et d'une méthode (protégée) `notifyAll(...)` permettant de signaler un changement à tous les observateurs abonnés.
  Cette classe est destinée à être étendue pour contenir les données observées avec leurs méthodes de lecture et de modification, ces dernières faisant appel à `notifyAll(...)` dans le cas où il y a eu un changement de la valeur stockée.
- La classe `Observer` contient une simple méthode virtuelle abstraite `update(...)`, qui sera appelée par `Observable::notifyAll(...)`.

Exercice 1 Programmez les classes `Observable` et `Observer` décrites ci-dessus.

- Comme vous ne savez pas encore quelles informations seront passées à `update(...)`, programmez des classes génériques, dont le paramètre de type sera le type d'un des arguments (peut-être le seul) de `update(...)`.
- Si on ne souhaite pas que `Observable` se cantonne uniquement à l'observation d'un attribut, on peut, à la place, déclarer une méthode abstraite dont la valeur de retour sera passée à `update()`.
- Réfléchissez bien à ce qu'on met dans la liste des abonnés. Comme on ne veut pas dupliquer les observateurs, il faut des références ou des pointeurs. Mais attention à ce qui pourrait se passer après la destruction d'un observateur. Possibilités : utiliser `weak_ptr`, ou bien ajouter le désabonnement à la destruction d'un observateur.
- Pour tester, étendez ces classes pour créer un observable contenant un entier et un observateur qui affiche la nouvelle valeur entière quand on appelle sa fonction `update(...)`. Testez dans un `main()` approprié.

Remarque : pour l'instant, sans cas d'application réel, il est difficile d'être certain que ces deux classes fournissent exactement les fonctionnalités nécessaires. Il faut donc se dire qu'il s'agit juste d'une implémentation provisoire, peut-être à modifier dans la suite.

Exercice 2 On programme maintenant une application d'enchères en ligne.

La particularité de cette application, c'est la possiblité, pour un utilisateur, de démarrer un robot qui sur-enchérit automatiquement, pour un objet donné et dans une limite fixée, dès que quelqu'un d'autre a la meilleure enchère.

Le patron observateur/observable apparaît ici : l'observable, c'est la valeur de la dernière enchère sur l'objet en vente, les observateurs, ce sont les robots.

Pour ce qui est de la partie visible du progamme, écrivez une boucle **while** principale, dans laquelle l'utilisateur-testeur peut entrer des commandes pour :

- mettre en vente un objet (avec prix initial)
- faire une enchère (manuelle) pour un acheteur donné, sur un objet donné
- créer une enchère automatique, pour un acheteur donné, sur un objet donné, avec le montant maximal donné
- annuler une enchère automatique
- déclarer la fin d'une enchère (le nom du vainqueur et le prix de vente final doivent alors s'afficher).

Pour que ce soit pratique, dans cette interface utilisateur, on pourra identifier les objets et les utilisateurs par leurs noms (chaîne de caractères).

Faites en sorte qu'un message s'affiche quand la valeur de la dernière enchère sur un objet en vente change (on affiche le prix proposé et la personne qui l'a proposé).