Université de Paris
Master 1

Langages à Objets Avancés
Année 2021—2022

TP n°5

Amitié entre classes

La première section de ce sujet porte sur les listes chaînées et était déjà présente sur la feuille précédente. Peut être l'avez vous traité en exercice à la maison ; si c'est le cas vous pouvez aborder la seconde section.

Listes doublement chaînées

On rappelle qu'on implémente les listes en utilisant deux classes : les cellules, et une encapsulation de cellule (ce qui permet de définir la liste vide l'identifier à `nullptr`)

Lorsque la liste est dite doublement chaînée, les cellules sont composées de trois champs : son contenu, un pointeur vers la cellule précédente et un pointeur vers la cellule suivante. Ces pointeurs sont `nullptr` en cas d'absence de précédent ou de suivant. [1]

Ces champs seront évidemment encapsulés et cachés au monde extérieur, qui ne pourra accéder à la liste qu'au travers d'un certain jeu de méthodes garantissant que la liste préserve une structure cohérente.

On se focalisera ici sur les listes chaînées d'entiers.

Exercice 1 [Cellule]

1. Écrire la classe `Cell`.

   Cette classe contient, outre les 3 champs déjà mentionnés, un constructeur adéquat, une méthode `connect` permettant de connecter deux cellules (pensez à modifier le champs `next` de l'une et `previous` de l'autre) et les méthodes `disconnect_next` et `disconnect_previous` (idem : pensez à mettre à jour l'ancienne cellule voisine).

2. Si on veut faire jouer un rôle symétrique aux deux cellules que l'on connecte, en permettant un appel de la forme `Cell::connect(c1, c2)` (au lieu de `c1.connect(c2)`), quelle sera la déclaration correcte de cette méthode ?

3. Faites en sorte que le monde extérieur ne puisse pas modifier des cellules de façon incohérente (notamment, pour toute cellule `c`, il faut que la cellule précédente de la suivante de `c` soit toujours `c`). Pour cela, jouez sur les modificateurs de visibilité (`private`) et ajoutez des accesseurs en lecture seule s'il le faut.

Exercice 2 [Liste]

On écrit maintenant la classe `List` qui doit fournir les méthodes usuelles :

— `int length()` : longueur de la liste ;
— `int get(int idx)` : valeur du `idx`-ième élément de la liste ;
— `int find(int val)` : indice de la valeur `val` si elle existe dans la liste, `-1` sinon ;
— `void set(int idx, int val)` : affecte la valeur `val` à la position `idx` de la liste ;
— `void insert(int idx, int val)` : insère la valeur `val` en position `idx` (et décale les éléments qui suivent) ;

1. Vous pourrez vous interroger sur les raisons qui font que l'on utilise des pointeurs et non des références.

— **void delete(int idx)** : supprime la valeur d'indice `idx` (et décale les éléments qui suivent).

1. Écrivez la classe `List`, munie de champs privés pointant sur la première et la dernière de ses cellules, d'un constructeur instanciant une liste vide, un destructeur qui désalloue les cellules de la liste et les méthodes mentionnées ci-dessus.
2. Ajustez l'encapsulation de la classe `Cell`, afin que seule la classe `List` puisse instancier et manipuler des cellules (qui ne sont qu'un intermédiaire technique pour implémenter une liste chaînée et n'ont pas vocation à être visibles pour les autres classes).

   Indice : il faudra utiliser **private** et **friend**.
3. Ajoutez un constructeur de copie sur les listes. Avez vous pensé à ce qu'il se passe lors de l'affectation entre listes ? (c. à d. lorsque vous écrivez `l1 = l2`)
4. Testez toutes les méthodes ! Comment peut-on faire pour tester les valeurs des champs et méthodes privés, et malgré tout regrouper tous les tests dans une classe séparée ?

Voici venu le temps des élections

Exercice 3 On souhaite modéliser un scrutin pour des élections. Pour un scrutin donné, on gère plusieurs bureaux de vote (avec dans chacun une urne). Le nombre d'options de vote possibles change à chaque scrutin : par exemple, pour un référendum, il y a 3 options "oui", "non" et "vote nul ou blanc".

On va avoir une classe `Scrutin` qui contiendra le nombre de bureaux de Vote (et donc d'urnes), le nombre d'options de vote et un tableau de pointeurs sur les urnes. On fera aussi une classe `Urne` qui contiendra une référence sur `Scrutin`, un entier représentant le numéro du bureau de vote (utilisez un compteur « static ») et un tableau d'entier comptabilisant les votes pour chaque option.

De plus, `Urne` aura une méthode `bool voter(int choix)`, qui retournera `false` si l'option est impossible, et vous ajouterez les méthodes nécessaires pour pouvoir obtenir les résultats d'un bureau de vote, de celui du scrutin entier et d'afficher ces résultats.

Indication : Vous avez dû remarquer qu'une urne contient une référence à un scrutin qui lui contient un tableau d'urnes. Si vous essayez de mettre `#include "Scrutin.hpp"` dans le fichier Urne.hpp, et vice-versa, le compilateur refusera.

Pour résoudre le problème, dans le fichier Urne.hpp, on déclare `class Scrutin;` avant la déclaration de la classe `Urne` et on fait un `#include "Urne.hpp"` dans Scrutin.hpp. La déclaration `class Scrutin;` suffit car, dans la déclaration de la classe `Urne`, on n'utilise pas d'autre information que le fait que cette classe existe.

Exercice 4 Écrire les destructeurs des classes `Urne` et `Scrutin`. À la fin d'un scrutin, on détruit les urnes (dans la réalité leur contenu). Vérifiez avec des sorties écran appropriées que l'on détruit bien les urnes.

Exercice 5 Pour éviter qu'on puisse fabriquer des urnes et les rattacher à un scrutin indûment. On va rendre les constructeurs et destructeurs d'`Urne` privées. Pour que `Scrutin` puisse construire des Urnes et les détruire, on va déclarer la classe `Scrutin` amie de `Urne` en écrivant dans la déclaration de classe de `Urne` : `friend class Scrutin;` Cette déclaration d'amitié va permettre à `Scrutin` d'utiliser les membres de `Urne` qui ne sont pas publiques.

Exercice 6 Ajoutez des `const` partout où c'est possible : attributs, méthodes, ...

Pour aller plus loin

Exercice 7 Comment éviter que l'on puisse voter après la fin du scrutin et que l'on puisse afficher les résultats avant la fin du scrutin ?

Exercice 8 Pour l'instant, on ne contrôle pas qui vote et combien de fois. Proposez une modélisation plus complexe qui résolve le problème.