Université Paris Diderot **Langages à objets avancés**
M1 **Année 2016-2017**

TP n° 5 : pointeur, static, const

Motif Singleton

Exercice 1 On veut implémenter le motif (pattern) singleton. Ce motif sert pour obtenir une classe qui a au plus une instance. Il est utile, par exemple, pour faire une classe d'écriture sur la sortie standard, pour gérer des préférences, ...

Créez une telle classe `Single` qui contiendra un entier avec ses acesseurs : L'idée est de déclarer un attribut static `element` contenant un pointeur vers un objet de type Single. Une méthode statique `Single * getInstance()` qui, si `element` est nul, crée un nouvel objet `Single` et l'affecte à `element` ; et dans tous les cas retourne cet attribut.

Comment empêcher la création directe d'un `Single`. Peut-on envisager que l'attribut soit une reférence sur `Single` ? Que doit-on faire du destructeur et du constructeur de copie ?

Que ce passe-t'il en cas d'affectation du type :

```
Single* s1 = Single::getInstance();
Single s2 = *s1.
```

Voici venu le temps des élections

Exercice 2 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, 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 et un tableau d'entier conptabilisant 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 3 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 4 Pour éviter qu'on puisse fabriquer des urnes et les rattacher à un scrutin indûment. On va rendre les constructeurs et destructeurs d'`Urne` private. 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 5 Ajoutez des `const` partout où c'est possible : attributs, méthodes, ...

Pour aller plus loin

Exercice 6 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 7 Pour l'instant, on ne contrôle pas qui vote et combien de fois. Proposez une modélisation plus complexe qui résolve le problème.