Programmation Orientée Objet (C++) : Collections hétérogènes

Jamila Sam

Laboratoire d'Intelligence Artificielle
Faculté I&C

Vidéos, quiz et transparents

https ://www.coursera.org/learn/programmation-orientee-objet-cpp/home/week/5

- ☞ Semaine 5 (partie avec les collections hétérogènes)

Jamila Sam
& Jean-Cédric Chappelier

Collections hétérogènes en pratique

Il y a en fait de nombreuses conceptions possibles pour les collections hétérogènes, principalement suivant deux axes :

① Le contenu de la collection est-il personnel (à la collection) ou partagé ?

☞ **QUI** a la propriété des objets dans la collection : la collection elle-même ou un responsable externe ?

Il est conseillé ici :

- Tant que faire se peut de donner la propriété à la collection
- et dans ce cas d'utiliser des `unique_ptr` en C++11 ou des pointeurs à la C, mais gérés en interne de la collection
- sinon (si vous ne pouvez pas donner la propriété à la collection), d'utiliser des pointeurs à la C et transférer des adresses d'objets existants plus longtemps que la collection elle-même
  (il faudra donc le garantir !)

Collections hétérogènes en pratique (2)

Il y a en fait de nombreuses conceptions possibles pour les collections hétérogènes, principalement suivant deux axes :

② **quelle interface** pour les utilisateurs de la classe ?

2.a) Gérer les pointeurs en interne (après tout ce n'est qu'un détail d'implémentation lié au langage) :

```
void Collection::ajoute(un_type const&);
```

voire

```
void Collection::ajoute(un_type&);
```

2.b) ou alors « afficher » les pointeurs et laisser leur gestion en externe :

```
void Collection::ajoute(un_type*);
```

La première (2.a) est plus « propre » (cache les détails d'implémentation), la seconde (2.b) plus directe (et plus simple pour le programmeur de la classe).

Dans le premier cas, il pourrait peut être s'avérer utile d'avoir une copie polymorphique des éléments contenus :

```
virtual un_type* un_type::copie() const;
```

Collections hétérogènes en pratique (3)

Dans le second cas (2.b, gestion externe des pointeurs) :

```
void Collection::ajoute(un_type*);
```

on peut, en tant qu'*utilisateur* de la collection, opter pour une version statique :

```
un_type_possible un_element(...);
...
collection.ajoute(&un_element);
```

(mais attention à la portée des variables : ne pas avoir une collection de plus grande portée que ses éléments !

ou alors dynamique :

```
collection.ajoute(new un_type_possible(...));
```

Dans tous les cas (gestion interne ou externe dynamique), il ne faut pas oublier de gérer correctement la libération de la mémoire.

Pour préparer le prochain cours

- Vidéos et quiz du MOOC semaine 6 :
  - Héritage multiple : concept et constructeurs [11 :14]
  - Héritage multiple : masquage [7 :06]
  - Classes virtuelles [15 :02]

- Le prochain cours :
  - de 14h15 à 15h (compléments)

Jamila Sam
& Jean-Cédric Chappelier