Programmation Orientée Objet : Constructeurs/Destructeurs

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs de la leçon d'aujourd'hui

- Concepts fondamentaux
- Compléments non abordés dans le MOOC
- Étude de cas

Organisation du travail (semestre)

		MOOC	déc.	cours 1 h Jeudi 8-9	exercices 2 h Jeudi 9-11
1	22.02.24		0	Intro + compil. séparée	
2	29.02.24	1. Intro POO	0	Intro POO	
3	07.03.24	2. Constructeurs/Dest	0	Constructeurs	
4	14.03.24	3. Surcharge des opé	0	Surcharge	
5	21.03.24	4. Héritage	0	Héritage	
6	28.03.24	5. Polymorphisme	0	Polymorphisme 1	
-	11.04.24		-	vacances Pâques	
7	04.04.24		1	Polymorphisme 2 / Collections hétérogènes	
8	18.04.24		-	-	Série notée
9	25.04.24	6. Héritage multiple	2	Héritage multiple	
10	02.05.24	(7. Etude de cas)	-	Templates	
12	16.05.24		-	(Ascension)	
11	09.05.24		-	Structure de données abstraites ; Bibliothèques	
13	23.05.24	(7. Etude de cas)	-	Bibliothèques (fin) + Révisions	
14	30.05.24		-	-	Examen

Concepts fondamentaux

- Rôles des constructeurs :
  initialiser les instances
- Ecriture des constructeurs :
  utilisez la liste d'initialisation, « section deux-points »
- Ecriture des constructeurs :
  il n'est très souvent **pas** nécessaire d'écrire le constructeur de copie
  (pensez alors au destructeur et à `operator=`)
- Rôles des destructeurs :
  faire, *si nécessaire*, ce que l'on doit faire *avant* la disparition de l'instance

C++11 Initialisation avec des listes

C++11 a généralisé la notion de listes de valeurs.

Nous avions par exemple vu :

```
vector<int> ages({ 20, 35, 26, 38, 22 });
```

Vous pouvez aussi munir vos classes d'une construction par listes avec le constructeur suivant :

```
NomClasse(initializer_list<type> const&)
```

(nécessite un `#include <initializer_list>`)

Si nécessaire, parcourez la liste avec un « *range-based for* » (ou utilisez le constructeur d'une autre classe qui supporte les listes d'initialisation, telles que `vector` ou `array`).

Exemple :

```
class A {
public:
  A(initializer_list<double> const& liste) {
    for (auto a : liste) { cout << a << endl; }
  }
};
```


C++11 Constructeur de déplacement

C++11 fournit également un moyen de gérer les instances temporaires (« *r-values* ») : le *constructeur de déplacement* (« *move constructor* »)

Ce constructeur permet d'initialiser une instance en *déplaçant* les attributs d'une autre instance, temporaire, du même type.

Syntaxe :

```
NomClasse(NomClasse && autre)
: ...
{ ... }
```

Exemple (pas très pertinent) :

```
Rectangle(Rectangle && autre)
  : hauteur(move(autre.hauteur))
  , largeur(move(autre.largeur))
{}
```

Note : l'exemple ci-dessus n'est pas très pertinent car la classe en question est petite. L'intérêt est ici limité. Gérer la « *move semantics* » prend plus de sens pour des classes ayant (potentiellement) de **gros contenus** (typiquement au travers d'un *pointeur*).

Et c'est de toutes manières un sujet avancé !

Etude de cas

Comment initialiser nos nombres complexes ?

Première idée : parties réelle et imaginaire

Ensuite :

1re question : constructeur par défaut ?

- En **a**-t-on un ?
- En **veut**-on un ?

2e question : plongement des réels ?

3e question : constructeur de copie ?

4e question : autres constructeurs ? (ρ, θ) ?

Premier constructeur

Idée la plus simple : par coordonnées cartésiennes

```
class Complexe {
public:
  // constructeurs
  Complexe(double abscisse, double ordonnee)
  : x_(abscisse),  y_(ordonnee)
  {}
  //...
};
```

Utilisation :

```
Complexe z1(1.1, 2.2);
```

Constructeur par défaut ?

A-t-on déjà un constructeur par défaut ?

Peut-on écrire

```
Complexe z2;
```

?

Veut-on un constructeur par défaut ?

Plongement des réels ?

Comment faire pour « plonger $\mathbb{R}$ dans $\mathbb{C}$ » ?

```
Complexe z3(1.0);
```


Constructeur de copie ?

Peut-on (déjà) construire des complexes comme ceci :

```
Complexe z4(z3);
```


Constructeur en coordonnées polaires ?

Peut-on construire des complexes comme ceci (pour $z_5 = e^{i\pi}$) :

```
Complexe z5(1.0, M_PI);
```

Question subsidiaire : si l'on décide de changer la *représentation interne* de nos nombres complexes (p.ex. en polaires), comment construire un nombre par coordonnées cartésiennes ?