

Programmation Orientée Objet : Polymorphisme, 1^{re} partie

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

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/Desi	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

Objectifs de la leçon d'aujourd'hui

- ▶ Concepts fondamentaux
- ▶ Étude de cas

Concepts fondamentaux

- ▶ notion de polymorphisme
 1. notion **FONDAMENTALE!**
 - ☞ Attention aux « tests de type » : **JAMAIS!!!**
 2. en C++ : 2 ingrédients : ...
- ▶ méthodes virtuelles :
 - `virtual` (transmis par héritage ; mais je vous conseille de le remettre à chaque fois pour mémoire)
 - `override` (optionnel, mais conseillé)
- ▶ méthodes virtuelles *pures* et classes *abstraites*
 1. `virtual plus = 0`
 2. on **ne** peut **pas** créer d'instance de classe abstraite !
- ▶ (semaine prochaine : collections hétérogènes)

Polymorphismes

En programmation, on distingue deux types de polymorphismes :

- ▶ le **polymorphisme des traitements** (ou ad hoc)
 - ☞ mécanisme de **surcharge** des fonctions/méthodes : le même identificateur est utilisé pour désigner des séquences d'instructions différentes
- ▶ le **polymorphisme des données** (ou universel)
 - ▶ le polymorphisme **d'inclusion** :
le même code peut être appliqué à des données de types différents liés entre eux par une relation de sous-typage
 - ☞ hiérarchies de classes
 - ▶ le polymorphisme **paramétrique** :
le même code peut être appliqué à n'importe quel type (généricité)
 - ☞ Cours sur les *templates* dans quelques semaines

Exemple illustratif : l'affichage polymorphique

```
class Affichable {
public:
    virtual void affiche(ostream& flot) const = 0;
};

// -----
ostream& operator<<(ostream& flot, const Affichable& objet)
{
    objet.affiche(flot);
    return flot;
}

// =====
class Machin: public Affichable
{
public:
    // ...
    virtual void affiche(ostream& flot) const override {
        // ...
    }
    // ...
};
```

Etude de cas

Série notée 2019
(sujet disponible sur le Moodle du cours)