

Programmation Orientée Objet (C++) Introduction

Jamila Sam

Laboratoire d'Intelligence Artificielle
Faculté I&C

©EPFL 2024-25
Jamila Sam
& Jean-Cédric Chappelier



Programmation Orientée Objet – Cours 16 : Introduction POO – 1 / 12

Concepts à voir durant le semestre

Objet	
Encapsulation et Abstraction Classes Héritage simple/multiple Polymorphisme Classes abstraites/virtuelles Résolution des collisions de noms	
Traitements	Données
Méthodes Constructeurs & Destructeurs Const Virtuelles (pures) Surcharge d'opérateurs(interne/externe) Privés/protégés/publiques Hérités/cachés (: :)	Attributs Appels aux constructeurs des attributs (hérités) Statiques

©EPFL 2024-25
Jamila Sam
& Jean-Cédric Chappelier



Programmation Orientée Objet – Cours 16 : Introduction POO – 3 / 12

Vidéos, transparents et quiz

<https://www.coursera.org/learn/programmation-orientee-objet/home/week/1>

Semaine 1

©EPFL 2024-25
Jamila Sam
& Jean-Cédric Chappelier



Programmation Orientée Objet – Cours 16 : Introduction POO – 2 / 12

Qu'est-ce que la Programmation Orientée Objets ?

Dans les grandes lignes :

- ▶ c'est une *manière de structurer les données et les traitements* interagissant dans un programme ;
- ▶ la POO n'est *pas spécifique à un langage particulier*
- ▶ en POO, un objet regroupe les données et les traitements relatifs à un concept particulier. Les objets interagissent entre eux.

Un objet peut être vu comme une *struct* :

- ▶ dotée de fonctions qui lui sont spécifiques (méthodes) et qui permettent de manipuler les données qu'elle contient ;
- ▶ dans laquelle l'accès direct aux données peut-être interdit pour définir un *cadre d'utilisation précis*.

©EPFL 2024-25
Jamila Sam
& Jean-Cédric Chappelier



Programmation Orientée Objet – Cours 16 : Introduction POO – 4 / 12

Concepts caractéristiques de la POO

- ▶ **Abstraction des données** : les données sont manipulées à l'aide de méthodes qui cachent leur représentation interne
 - ☞ on peut imposer que ces méthodes vérifient l'intégrité des données (*fiabilité*)
 - ☞ l'utilisation d'une donnée ne dépend plus des choix faits pour sa représentation (*facilité de maintenance*)
 - ☞ des données différentes peuvent être manipulées de manière identique (*concision*)
- ▶ **Héritage** : on peut prendre un type d'objets déjà existant et lui rajouter des données et/ou des méthodes pour en faire un nouveau type d'objets
 - ☞ on peut donc établir des liens sémantiques entre des types d'objets et éviter des redondances dans leurs descriptions (*concision*).

Quelques défauts aussi ...

La POO peut :

- ▶ impliquer une certaine lourdeur pour le codage de choses simples ;
- ▶ engendrer des coûts algorithmiques plus élevés pour la manipulation de types élémentaires

En C++ les types fondamentaux (élémentaires) ne sont pas des objets.

Avantages de la POO

- ▶ **Façon naturelle de modéliser** les données et traitements intervenant dans un programme
- ▶ **Modularité** : le programme est composé d'un ensemble d'entités (types d'objets) aux rôles bien déterminés
- ▶ **Réutilisabilité** : chaque type d'objets peut être réutilisé, sa sémantique peut être étendue par le biais de l'héritage
- ▶ **Polymorphisme** : un même code peut s'appliquer à des types d'objets différents
- ▶ **Abstraction** : la représentation des données est découplée de leur utilisation. De plus des règles précises d'utilisation des données peuvent être imposées.
- ▶ d'énormes bibliothèques déjà écrites (Java, C#)
- ☞ *Efficacité* dans la conception des programmes, *concision* du codage, meilleures *fiabilité* et *maintenabilité* des programmes

Exemples de langages orientés-objet

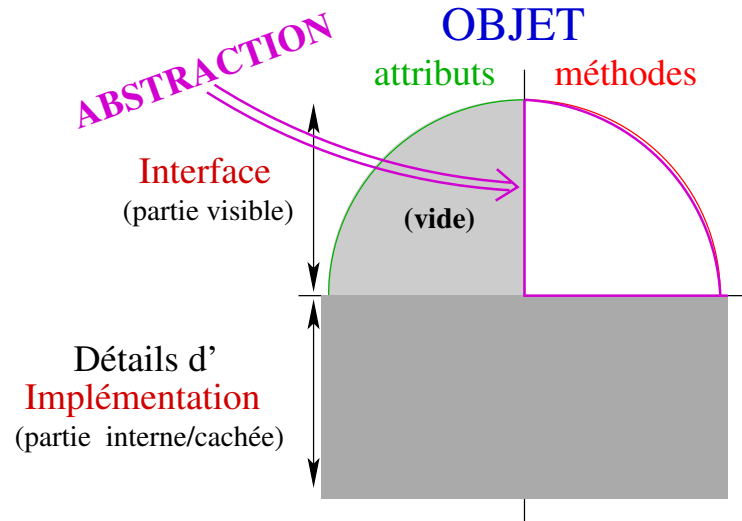
- ▶ Smalltalk (Xerox PARC, 1980) ;
- ▶ Objective-C (Apple, Brad Cox, 1986) ;
- ▶ Python (Open source, Guido van Rossum, 1990) ;
- ▶ Java (Sun Microsystem, 1995)
- ▶ C# (Microsoft, 2001)

Par ailleurs, la plupart des langages non-objets admettent une extension objets :

- ▶ Caml
- ▶ PERL
- ▶ PHP
- ▶ etc ...

Encapsulation / Abstraction : Résumé

MIEUX :



Etude de cas : modélisation d'un joueur...

- ▶ un joueur a un *pseudonyme* et un *temps de jeu* ;
- ▶ il peut gagner des *points* ;
- ▶ son score se calcule comme son nombre de points divisé par son temps de jeu

Classes et instances

```
class Concept {  
public:  
    methodes importantes;  
  
private:  
    attributs;  
  
    methodes secondaires;  
};  
  
Concept une_instance;
```

Pour préparer le prochain cours

- ▶ Vidéos et quiz du MOOC semaine 2 :
 - ▶ Constructeurs (introduction) [20 :02]
 - ▶ Constructeurs par défaut en C++ [21 :33]
 - ▶ Constructeur de copie [7 :54]
 - ▶ Destructeurs [14 :56]
- ▶ Le prochain cours :
 - ▶ de 14h15 à 15h (résumé et quelques approfondissements)