Université Paris Cité

# Compléments de POO

ECTS
3 crédits

Composante
UFR
Informatique

Période de l'année
Semestre 5

## En bref

- **Langue(s) d'enseignement:** Français
- **Méthode d'enseignement:** En présence
- **Organisation de l'enseignement:** Formation initiale
- **Forme d'enseignement :** Cours magistral & Travaux pratiques
- **Ouvert aux étudiants en échange:** Non

# Présentation

## DESCRIPTION

Ce cours de compléments de POO revisite les principes des langages objets, tels que vus en L2, avec une vision orientée vers la qualité, la pérennité (possibilité d'évoluer et de corriger les bugs) et la ré-utilisabilité des composants programmés.

Le langage qui sert d'illustration est Java. Pour chaque aspect de ce langage, tous les écueils probables sont mis en évidence et de bonnes pratiques permettant de les éviter sont expliquées (cela va du simple ajout de mot-clé tel que `private` ou `final`, à la mise en place d'un patron de conception complet).

Les ajouts récents du langage (Java 17 est la version actuellement utilisée pour le cours) sont expliqués quand il permettent d'améliorer la concision et la robustesse du code.

Un tiers du volume du cours est consacré à la programmation concurrente en Java, avec toujours l'idée de programmer de la façon la plus fiable, la moins risquée possible (on encourage a l'utilisation des API concurrentes de haut niveau plutôt que la manipulation directe des threads, quand c'est possible).

## PRÉ-REQUIS NÉCESSAIRES

* Savoir programmer des algorithmes et structures de données simples (n'importe quel langage).
* Avoir déjà programmé en Java.

## SYLLABUS

### Sujets centraux

* Qu'est-ce que la POO. Notions d'objet et de classe. Limitations des constructeurs # fabriques statiques, builders.
* Encapsulation (au sens large).
  * Notion de propriété (versus champs) avec getter/setter.
  * Problèmes liés à l'aliasing (références partagées).
  * Notion de copie défensive.
* Polymorphisme
  * différentes sortes.
  * Système de types.
  * Notion de contrat et de sous-typage idéal (Liskov) versus sous-typage implémenté. Illustration#: problème rectangle/carré.

**Pour en savoir plus, rendez-vous sur > u-paris.fr/choisir-sa-formation**

Université Paris Cité

    * Interfaces et programmation à l'interface (inversion de dépendance, adaptateurs, fabriques abstraites, … ).
* Héritage.
    * Héritage justifié seulement quand on veut sous-typage + réutilisation de code (sinon, autres constructions).
    * Problème de la classe de base fragile.
    * Comment contourner l'absence d'héritage multiple.
    * Hiérarchies contraintes : types finis (enum), hiérarchies scellées (sealed). Enregistrements (record). Évolutions du bloc de contrôle switch.
* Généricité.
    * principes de base
    * un cas d'étude :#les optionnels
    * lambda expressions
* Concurrence :
    * contexte : les applications sont de plus en plus concurrentes par nature (web, GUI, … ) et d'autre part le matériel (multicore) permet concrètement de plus en plus de parallélisme dont il est intéressant de profiter.
    * un peu de théorie : parallélisme versus concurrence. Temps partagé. Entrelacements. Conflit et compétition. Correction mise en danger par les optimisations matérielles (caches locaux) et logicielles (réordonnancement).
    * implémentation de base dans Java :#threads JVM (= simples décorateurs de threads système). Synchronisation (volatile, synchronized, wait/notify).
* Concurrence : APIs de haut niveau.
    * plusieurs styles : mémoire partagée versus passage de message (plus sûr !)
    * notion de thread pool
    * API ForkJoin
    * API CompletableFuture
    * Mention (ou rappel) des Stream parallèles. Mention de l'API Flow.

## Sujets potentiellement traités

Certains sujets ne sont pas systématiquement développés en amphi toutes les années (mais le cours est écrit et disponible)#:

    * Les détails des conversions de type (analyse du code-octet et du comportement de la JVM).
    * Gestion des erreurs. Exceptions et alternatives.
    * Rappel de la zoologie des collections Java.
    * Généricité avancée (explications plus théoriques sur l'effacement de type, les wildcards, la variance, la comparaison avec les tableaux)
    * API Stream.
    * Interfaces graphiques.