FORMATION

IT - Digital - Management

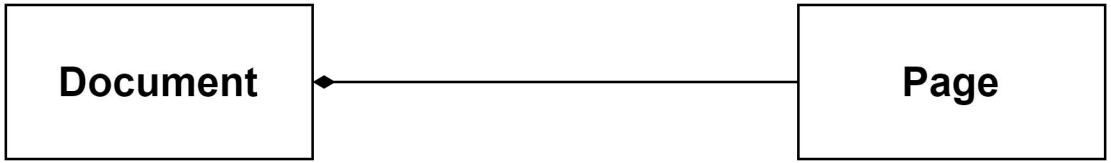
m2iformation.fr

Introduction à la POO

Néhémie “Alfred” BALUKIDI
CTO/Software Engineering & Data|ML/AI

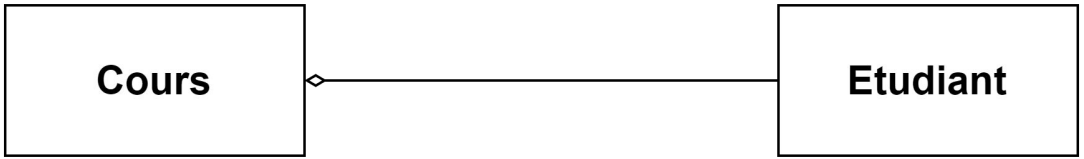
1. Un plus sur l'Héritage

Types d'Héritage: Composition



La composition est une association forte où un objet composite possède d'autres objets. Les objets composés n'ont pas d'existence indépendamment de l'objet composite. Lorsque l'objet composite est détruit, les objets composés le sont également.

Types d'Héritage: Agrégation



L'agrégation est une association permettant à un objet composite de référencer d'autres objets. Les objets référencés ont leur propre cycle de vie indépendamment de l'objet composite. Il s'agit d'une relation plus faible que la composition.

super()

Une méthode permettant d’accéder à la classe parente

super() permet d'accéder aux membres de la classe parente depuis une sous-classe

Ceci est utile pour :

- Appeler une méthode de la classe parente
- Accéder à des variables d'instance de la superclasse

Héritage multiple

Un langage autorise l'héritage multiple quando une classe peut hériter de plusieurs classes parentes

Utile pour :

- Factoriser des comportements communs
- Combiner des caractéristiques

Classe Abstraite

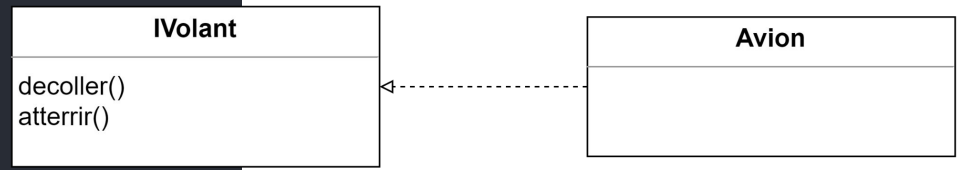
- Classe qui ne peut pas être instanciée
- Utile pour définir un "plan" commun à des sous-classes
- Contient généralement des méthodes abstraites
- Oblige les sous-classes à implémenter les méthodes abstraites

L’Interface

- Définit un « contrat » que des classes devront respecter
- Ne contient que des signatures de méthodes (pas d'implémentation)
- Permet de définir des comportements génériques
- Les classes implémentent l'interface et fournissent le corps des méthodes

```
interface IVolant {

    void decoller();
    void atterrir();
}
```



2. Les Design Patterns(Patrons de conception)

Qu’est-ce que c’est?

- Solutions éprouvées pour résoudre des problèmes courants dans la conception orientée objet
- Modèles de conception logicielle réutilisables
- Décrivent des relations et interactions types entre classes et objets
- Permettent de découpler les composants, promouvoir la flexibilité et la réutilisation

Objectifs des Design Patterns

- Réutilisabilité et flexibilité du code
- Limiter les dépendances et le couplage
- Faciliter l'évolution et la maintenance
- Implanter les principes SOLID

Exemples

- Fabrique (Factory)
- Singleton
- Observateur (Observer)
- Décorateur (Decorator)
- Stratégie (Strategy)
- Memento

3. Principes de conception

Les principes de conception OO

- Pas des dogmes, mais des lignes directrices
- Leur respect n'empêche pas les crashes, mais apporte d'autres bénéfices
- Parmi les plus connus : DRY, YAGNI, SOLID

SOLID

S : Single Responsibility Principle (SRP)

O : Open Closed Principle (OCP)

L : Liskov Substitution Principle (LSP)

I : Interface Segregation Principle (ISP)

D : Dependency Inversion Principle (DIP)

SOLID(cont.)

SRP - Principe de responsabilité unique

- Une classe doit avoir une seule responsabilité
- Changement localisé => Facilité de maintenance

OCP - Principe ouvert/fermé

- Ouvert aux extensions, fermé aux modifications
- Extensibilité par héritage et polymorphisme

SOLID(cont.)

LSP - Principe de substitution de Liskov

- Une sous-classe doit pouvoir remplacer sa super-classe
- Respect des contrats (pré et post conditions)
- Pas de renforcement des préconditions
- Pas d'affaiblissement des postconditions

SOLID(cont.)

ISP - Principe de ségrégation des interfaces

- Scinder les interfaces "grosses" en interfaces "petites"
- Client ne dépendre que des méthodes dont il a besoin
- Réduit le couplage et favorise la maintenabilité

DIP - Principe d'inversion de dépendances

- Dépendre d'abstractions plutôt que d'implémentations
- Découpler modules en utilisant interfaces et héritage
- Classes stables indépendantes des classes volatiles

FIN.