DDD
Domain Driven Design

Par Hugo Hermelin, Michel Wang, Yaniss Lasbordes, Noah Tilleul

Sommaire

1. Introduction

Qu’est ce que le DDD

- Méthode de conception où le logiciel répond aux besoins métier
- Comprendre ce que fait l’entreprise avant d’écrire du code

Collaboration experts métier / développeurs

- Les développeurs s’appuient sur les connaissances des experts pour bien comprendre les besoins

1. Introduction

Pourquoi utiliser le DDD

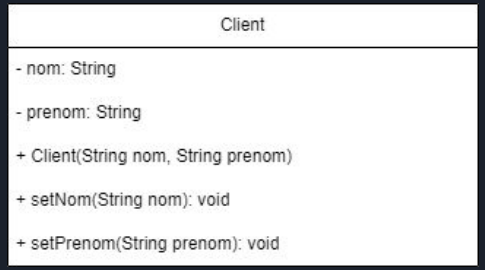
- Simplifier un système complexes en les divisant en petites parties
- Rendre le logiciel plus facile à comprendre / modifier / maintenir

Deux approches :

- Stratégique : Identifier les parties importante à traiter en priorité
- Technique : Utiliser des outils comme les entités et objets de valeur pour modéliser le domaine métier

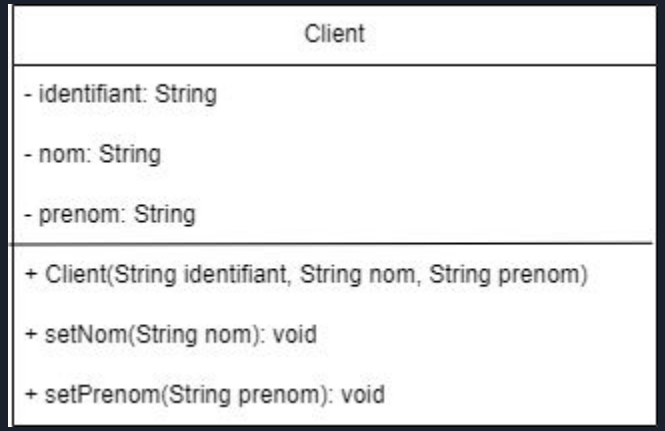
2. Entité (Entity)

- Ne peuvent pas être interchangés entre eux
- Ses attributs peuvent évoluer
- Les entités permettent de créer quelque chose d'unique et de distinguable parmi d'autre éléments



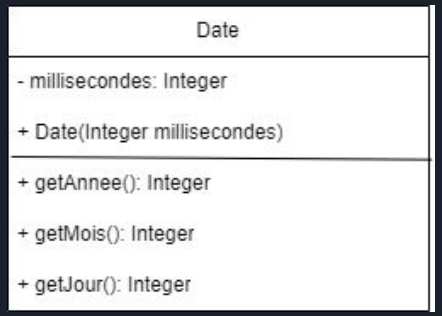
- Comment être sur que ce soit le même client tout en modifiant ses attributs ?

2. Entité (Entity)



3. Valeur object (Value Object)

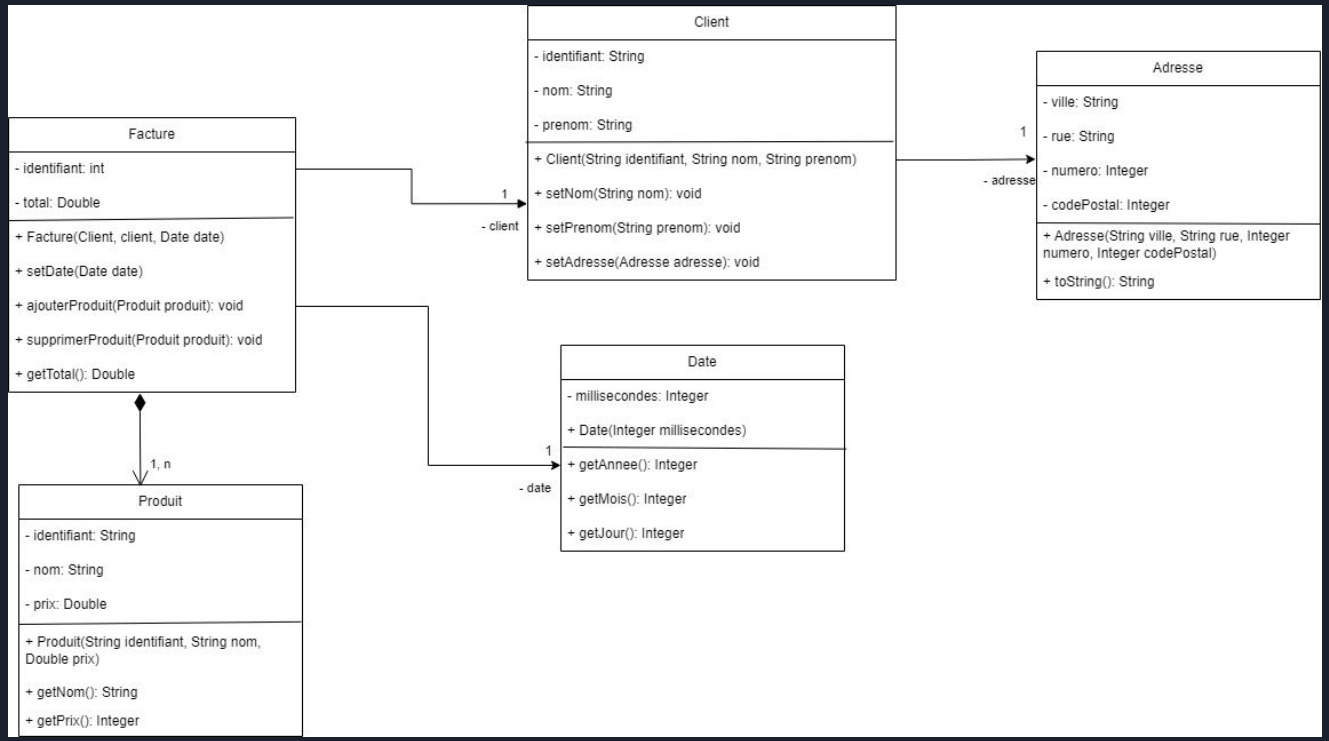
- Simplifient le design
- Partageables
- Copiables
- Pas d'existence propre
- Immuable
- Possède leur propre logique métier
- Comportement défini grâce à l'ensemble de ses attributs



4. Agrégat (Aggregate)

- Ensemble d’objets liés
- Traités comme une seule et même unité
- Possède une **racine** (root) qui contrôle l’accès à la modification
- Toutes actions doivent passer par lui
- Garantissent la cohérence de l’objet
    - Responsable des règles de validation

4. Agrégat (Aggregate)

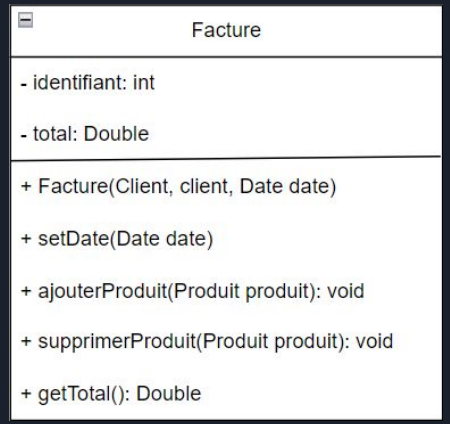


5. SRP : Principe de responsabilité unique

Définition : Chaque classe doit avoir une seule responsabilité

La classe « Facture » doit :

- Gérer l'ajout et la suppression de la liste de produits
- Calculer le montant total de la facture



Solution : Découper la classe Facture en plusieurs classes

- GestionProduit
- CalculTotal

5. OCP : Principe d’ouverture et fermeture

Définition : Une classe doit être ouverte aux extensions mais fermée aux modifications

Respecté ? :

- Modification de la classe « Facture », si on veut ajouter de nouvelles fonctionnalités

  Ex : la limitation du nombre maximal de clients

Solution : Utiliser une interface

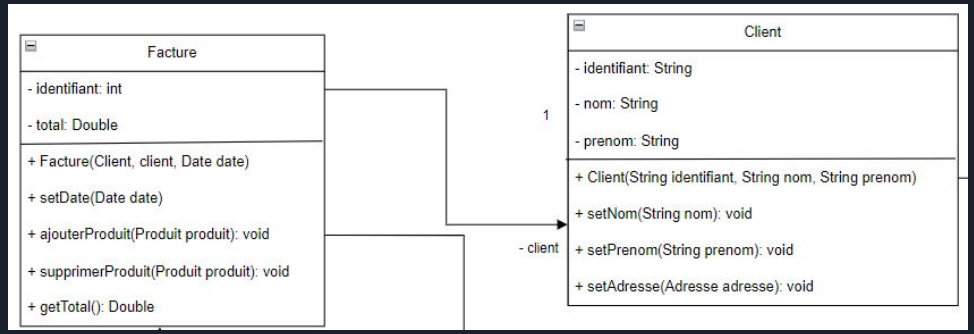
- Validation (interface)

5. LSP : Principe de substitution de liskov

Définition : Les sous-types doivent pouvoir être substitués à leur type de base

Respecté ? :

- Pas d'héritage entre les entités



5. ISP : Principe de ségrégation des interfaces

Définition : Favoriser l'implémentation de plusieurs interfaces au lieu d'une seule

Respecté ? :

- Pas d'interface

5. DIP : Principe d'Inversion des Dépendances

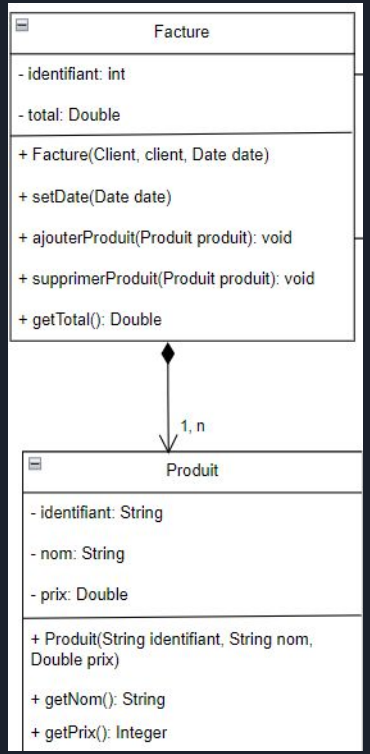
Définition : Il faut dépendre des abstractions et pas des implémentations

Respecté ? :

- Si on décide de rajouter un autre type de produit comme « ProduitPomotionnel » ou « PackProduit », on devra modifier la structure de « Facture »
- La classe « Facture » dépend donc de la classe « Produit »

Solution :

- Utiliser une interface « InterfaceProduit »



5. Liens principes SOLID

Principes respectés :

- LSP
- ISP

Principes non-respectés :

- SRP
- OCP
- DIP

6. Les limites du DDD

- Efforts conséquents nécessaires :

Le pattern peut apporter une grande complexité et peut donc ne pas être pertinent selon la taille du projet

- Nécessite une bonne connaissance du métier sujet :

Puisque le langage métier est au centre de ce pattern, il faut faire appel à des experts-métier pour s'aider (Il peut ensuite aussi y avoir un risque de mauvaise communication/compréhension)

7. Lien avec le pattern Immuable

Le pattern Immuable : Rien dans le projet ne doit pouvoir être modifié (pas de setter, tout est fait dans le constructeur), la seule alternative est de créer de nouvelles instances pour chaque modifications voulues.

- Lien avec les Entités :

Objets non interchangeables qui possèdent un existence propre notamment par exemple grâce à un id unique et immuable, sans rendre le reste de l'objet immuable.

- Lien avec les Objets de Valeur :

Objets complètements immuables dont tout est défini au constructeur

8. Live Coding

https://youtu.be/_9pvESpzz50

9. QCM

10. Sources

- https://alexsoyes.com/ddd-domain-driven-design
- https://chatgpt.com/
- https://www.jdecool.fr/blog/2020/10/18/la-notion-d-agregat-en-ddd.html
- https://github.com/iblasquez/enseignement-but2-developpement/blob/master/cours/SOLID.pdf