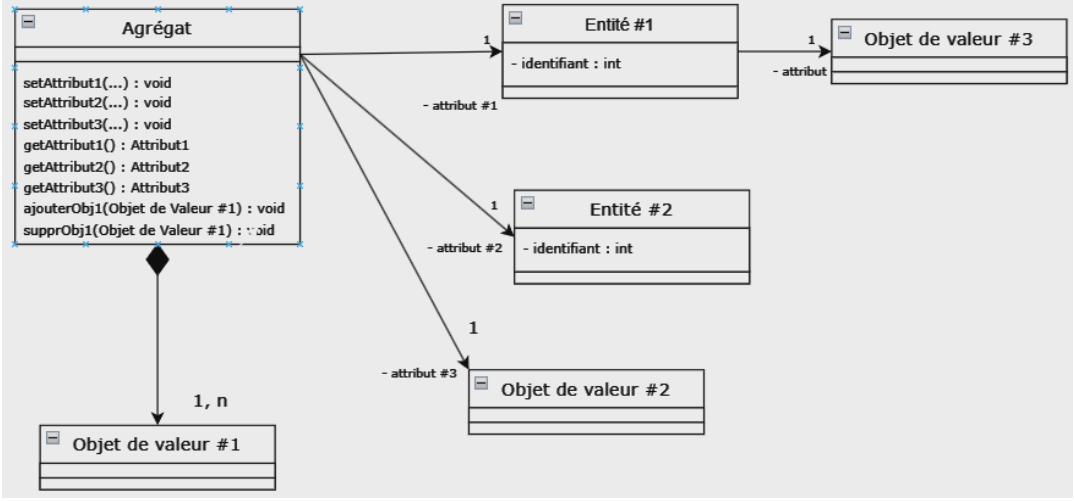
Domain Driven Design

Le DDD est un modèle de conception pilotée par le domaine qui n'est en soit pas un Design Pattern du GoF. Ce modèle se focalise sur le nommage précis des variables métiers dans un projet afin d'aider les équipes à produire et maintenir des solutions logicielles pour notamment des métiers fonctionnellement complexes en coupant en sous-domaine pour séparer les responsabilités.

Il se caractérise par 3 éléments :

- Les objets de valeur sont des éléments partageables et copiables et peuvent valoir un autre objet de valeur. Ils sont Immuables, c'est-à-dire que tout est défini dans leur constructeur et n'ont pas de setters. (ex : une adresse reste la même)
- Les entités sont des éléments non interchangeables entre eux, et possèdent des attributs qui peuvent évoluer (ex : une personne vieillit)
- Les Agrégats sont des ensembles d'objets liés et traités comme une unité. Ils possèdent une racine qui contrôle l'accès aux modifications des objets qui en font partie. (ex : une facture possède une date, une personne étant un destinataire, une personne étant l'émetteur)

Exemple :



Ici, nous avons Entité #1 qui a comme attributs un identifiant et Objet de Valeur #3 qui est lui-même un attribut de l'Agrégat ayant aussi comme attributs l'autre Entité et deux objets de valeur. C'est par l'interface de l'agrégat que l'on peut accéder aux attributs et les modifier.

Principes SOLID respectés :

Liskov **S**ubstitution **P**rinciple : le DDD n'a rien de basé sur l'héritage de classes ce qui fait qu'il n'y a pas de raison qu'il entre en conflit avec ce principe.

Interface **S**egregation **P**rinciple : le DDD n'a rien de basé sur les interfaces non plus, ce qui fait qu'il n'y a pas de raisons qu'il entre en conflit avec ce principe non plus.