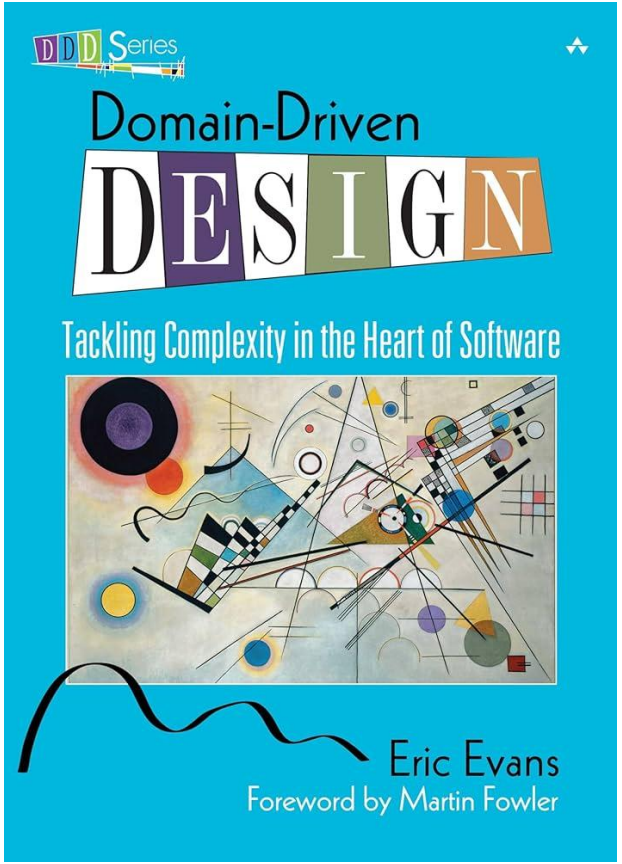
DOMAIN DRIVEN DESIGN
– DDD –

DIDRY Jules / HLADYSHEV Anton / PASTEL Gaël / GIANG Gia Bao

PLAN

1. Introduction
2. Entité
3. Valeur objet
4. Agrégat
5. Liens avec les principes SOLID
6. Limites du DDD
7. Live coding
8. QCM



Introduction

Définition

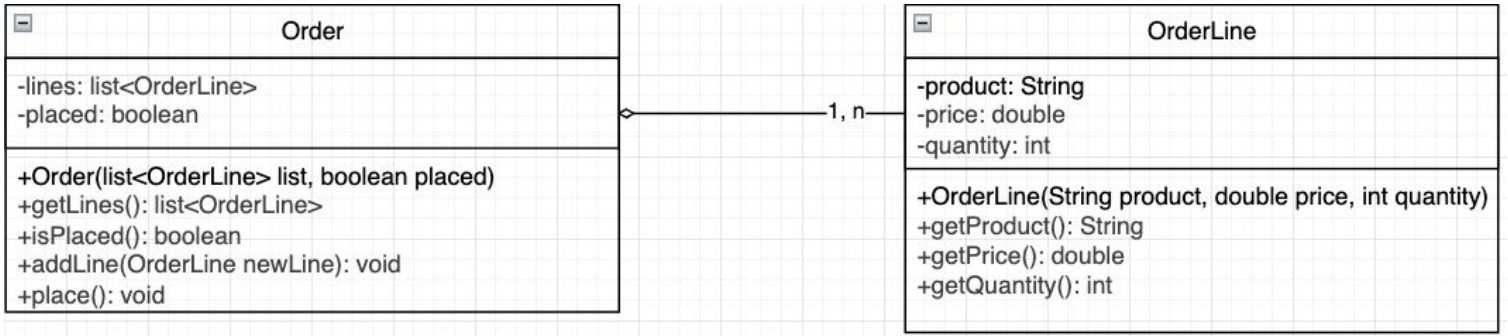
- Les patterns technique du DDD ne font pas partie des Design Patterns du **GoF**
- Une approche pour concevoir des logiciels complexes en se concentrant sur le domaine métier plutôt que sur la technologie ou l'infrastructure

Exemple métier - Un site d'e-commerce

Un client peut créer une commande, ajouter les articles, et ensuite la valider

```java
public class Order {
  private List<OrderLines> lines;
  private boolean placed;
}
```

```java
public class OrderLine {
  private String product;
  private double price;
  private int quantity;
}
```



Solution avec le DDD

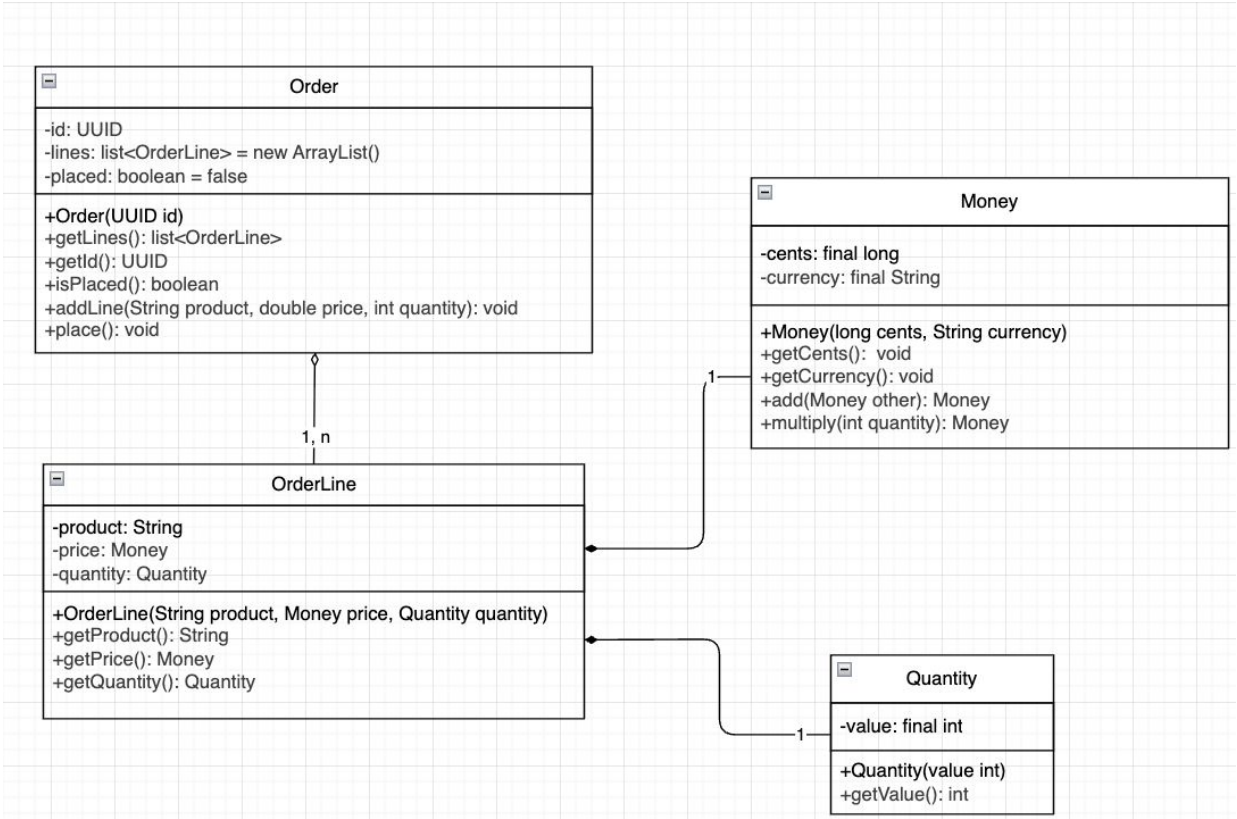
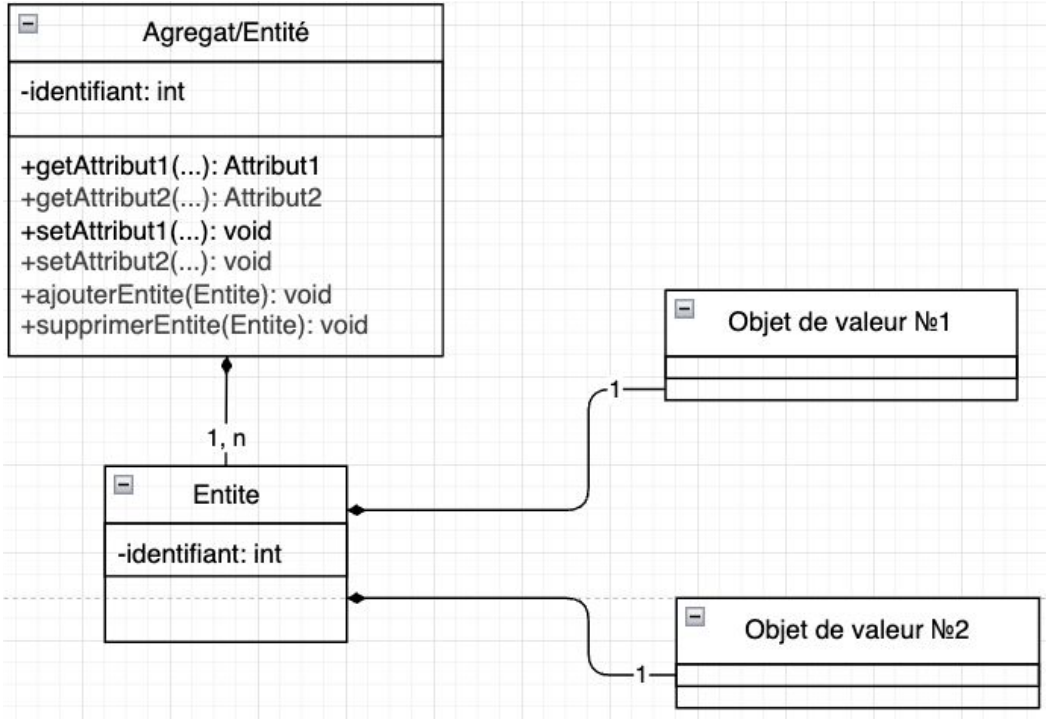


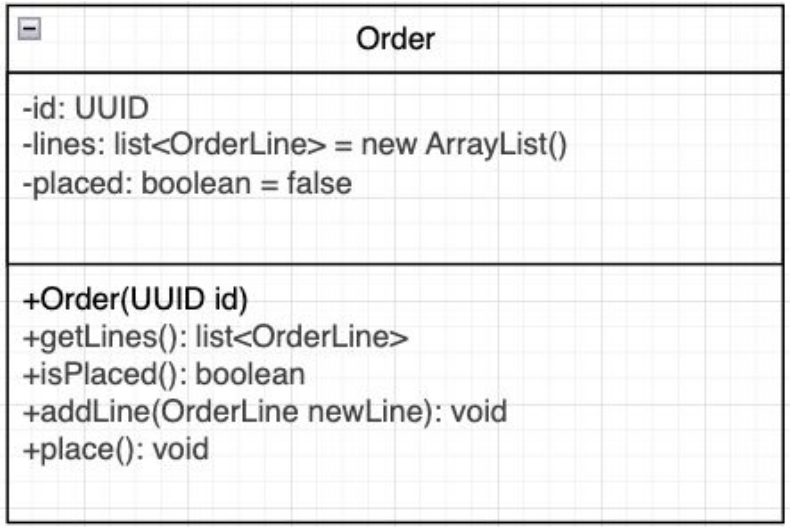
Diagramme Générique



Entity - Entité

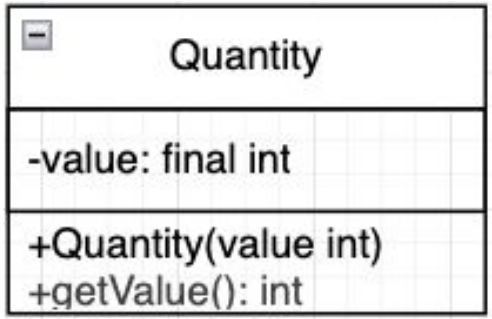
Une **entité** est un **objet métier** qui possède une **identité unique** et **persiste dans le temps**, même si ses attributs changent

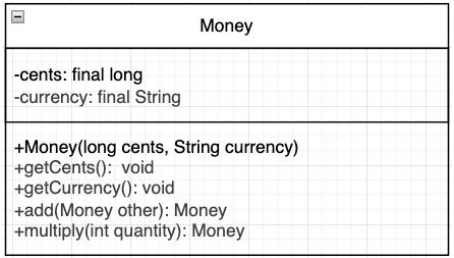
```java
public class Order {
    private final UUID Id;
    private final List<OrderLine> lines;
    private boolean placed;

    public Order(UUID Id) {
        this.Id = Id;
        this.lines = new ArrayList<>();
        this.placed = false;
    }
```



Value Object - Valeur Objet

Un **Value Object** est un objet **défini uniquement par ses valeurs, sans identité propre**.
Deux Value Objects sont **égaux si toutes leurs valeurs sont égales**.
Ils sont généralement **immuables** (on ne les modifie pas, on les remplace)





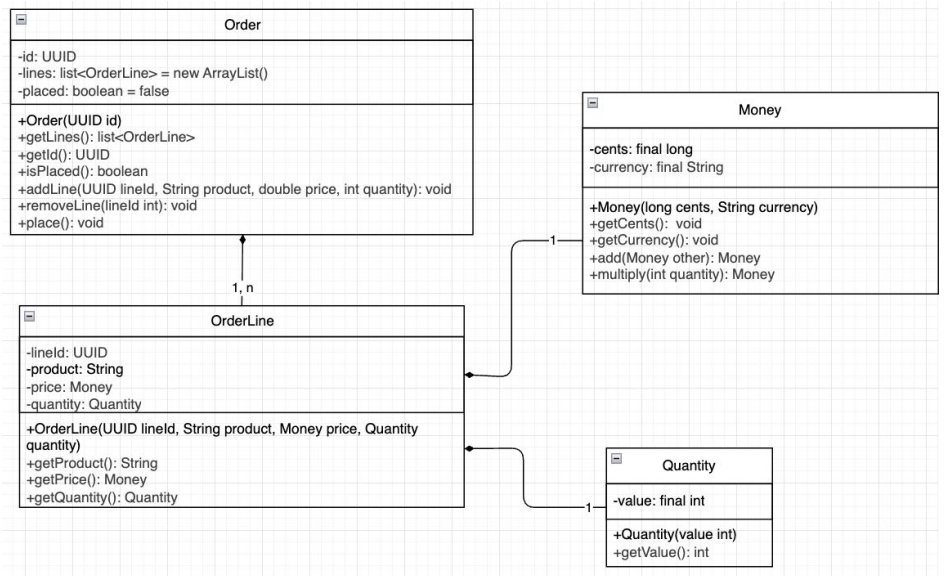
Value Object - Valeur Objet (record)

```java
public record Money(long cents, String currency) {

    public Money {
        Objects.requireNonNull(currency, "Currency must not be null");
        if (cents < 0) {
            throw new IllegalArgumentException("Amount cannot be negative");
        }
    }

    public static Money of(double amount, String currency) {
        long cents = Math.round(amount * 100);
        return new Money(cents, currency);
    }

    // Convertir en valeur décimale (ex: 1999 → 19.99)
    public double toAmount() {
        return cents / 100.0;
    }

    public Money add(Money other) {
        ensureSameCurrency(other);
        return new Money(this.cents + other.cents, this.currency);
    }

    public Money multiply(int quantity) {
        if (quantity < 0) {
            throw new IllegalArgumentException("Quantity cannot be negative");
        }
        return new Money(this.cents * quantity, this.currency);
    }
}
```

```java
public record Quantity(int value) {

    public Quantity {
        if (value <= 0) {
            throw new IllegalArgumentException("Quantity must be positive");
        }
    }

    public Quantity add(Quantity other) {
        return new Quantity(this.value + other.value);
    }
}
```

Aggregate - Agrégat

Un **agrégat** est un **ensemble cohérent d'objets du domaine** (entités et value objects)
regroupés autour d'une **entité racine** appelée **Aggregate Root**



Les classes participantes :

Classe	Rôle	Description
Entity	Modélise les **objets métier principaux** du domaine.	Porte une identité unique qui persiste dans le temps
Value Object	Modélise les **valeurs ou attributs métier**.	Décrit des caractéristiques sans identité, immuables
Aggregate	Assure la **cohérence du domaine**.	Regroupe entités et valeurs sous une racine unique contrôlant leur intégrité

Lien avec SOLID

Principes respectés naturellement :

- LSP → Pas d'héritage
- ISP → Pas d'interface
- OCP → Respecté
- DIP → Respecté

Principes non respectés : SRP

Les limites du DDD

- Complexité et surcharge de conception
- Nécessite une compréhension profonde du domaine
- Difficulté d’intégration dans des systèmes existants

Live Coding

QCM Interactif

Conclusion / Bibliographie

https://fr.wikipedia.org/wiki/Conception_pilotée_par_le_domaine

https://www.yieldstudio.fr/glossaire/domain-driven-design-ddd-modelisation

https://lesdieuxducode.com/blog/2019/7/introduction-au-domain-driven-design