

DDD Domain-Driven Design

GUILLON Benoît

GAILLARD Maxime

JAMALEDDINE Ali

BOULKRINAT Ilyès

Sommaire

Introduction générale sur les design patterns

Présentation rapide du DDD (stratégique-tactique)

Présentation de l'exemple utilisé

Value-Object

Entity

Aggregate

Vue d'ensemble de l'exemple

Conclusion

Annexes

Qu'est-ce qu'un design pattern?

En géométrie :

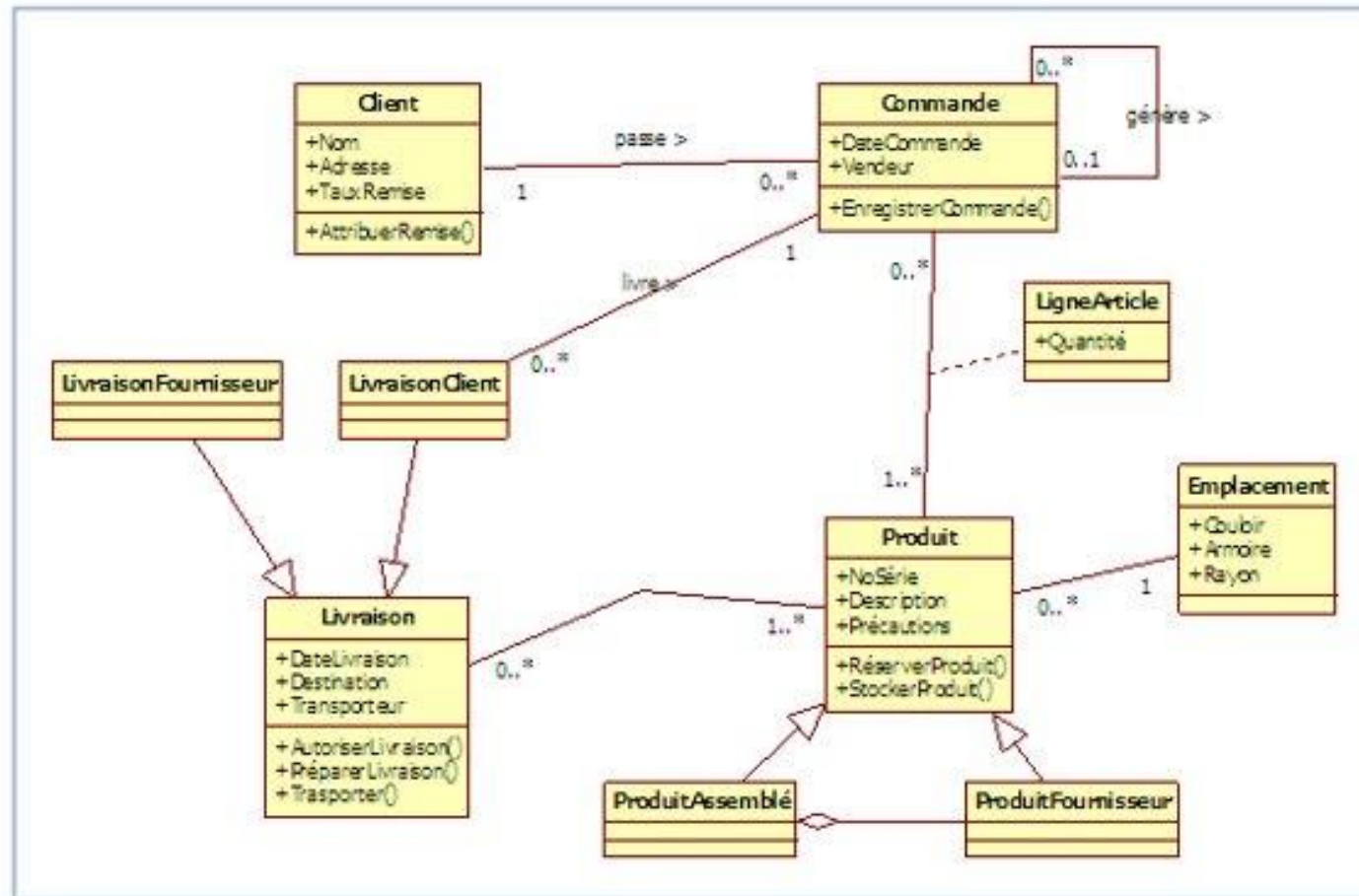


En POO:

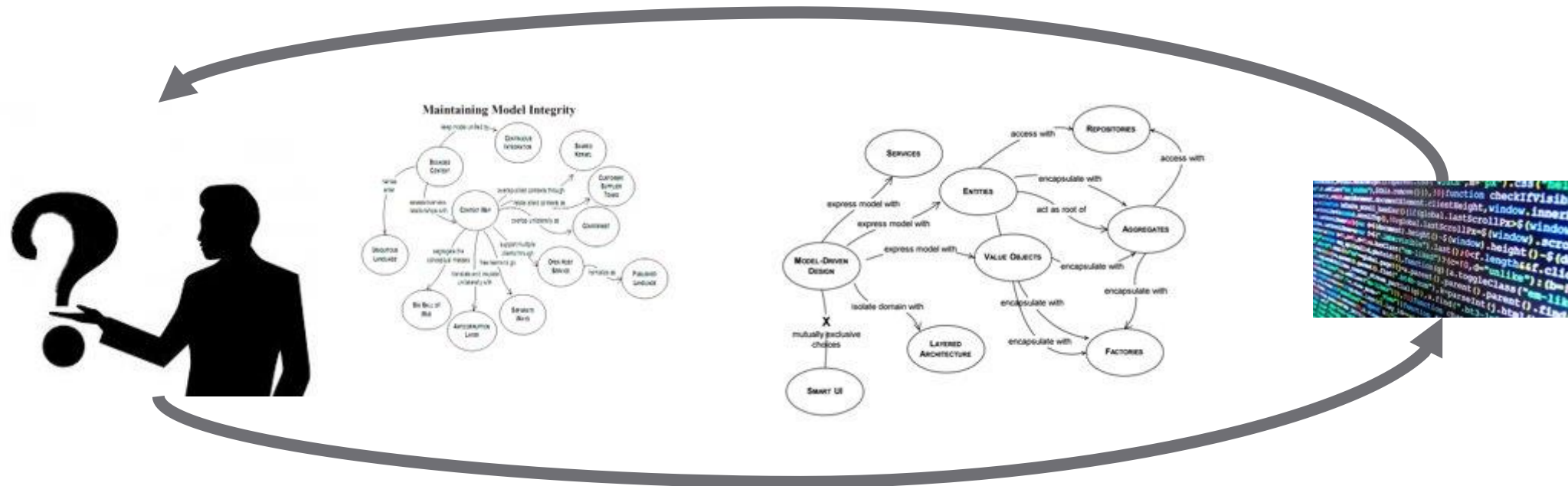
Meilleure solution au
problème

Problème récurrent

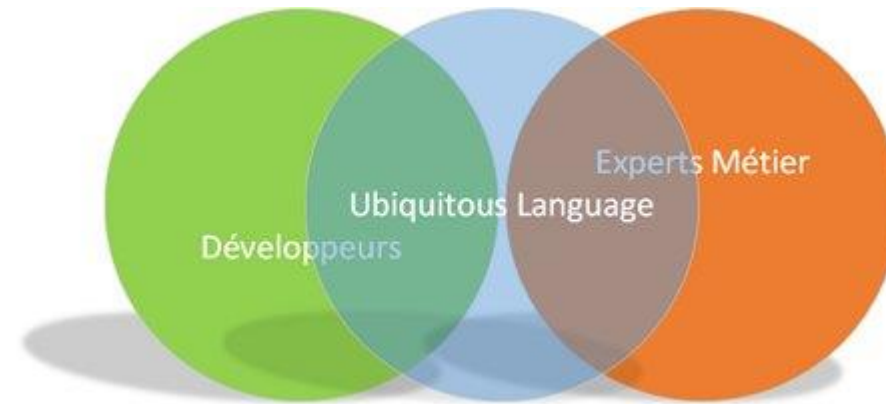
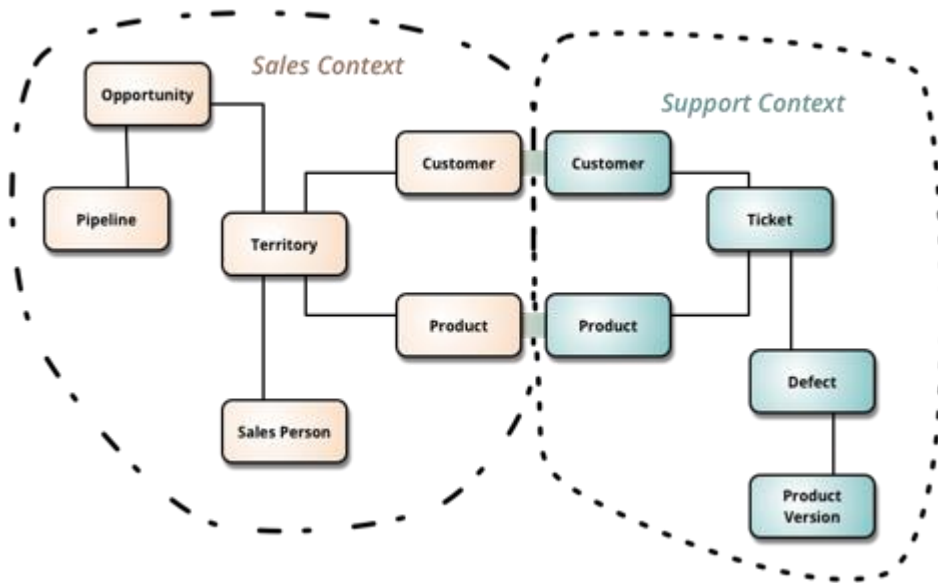
Qu'est-ce qu'un design pattern?



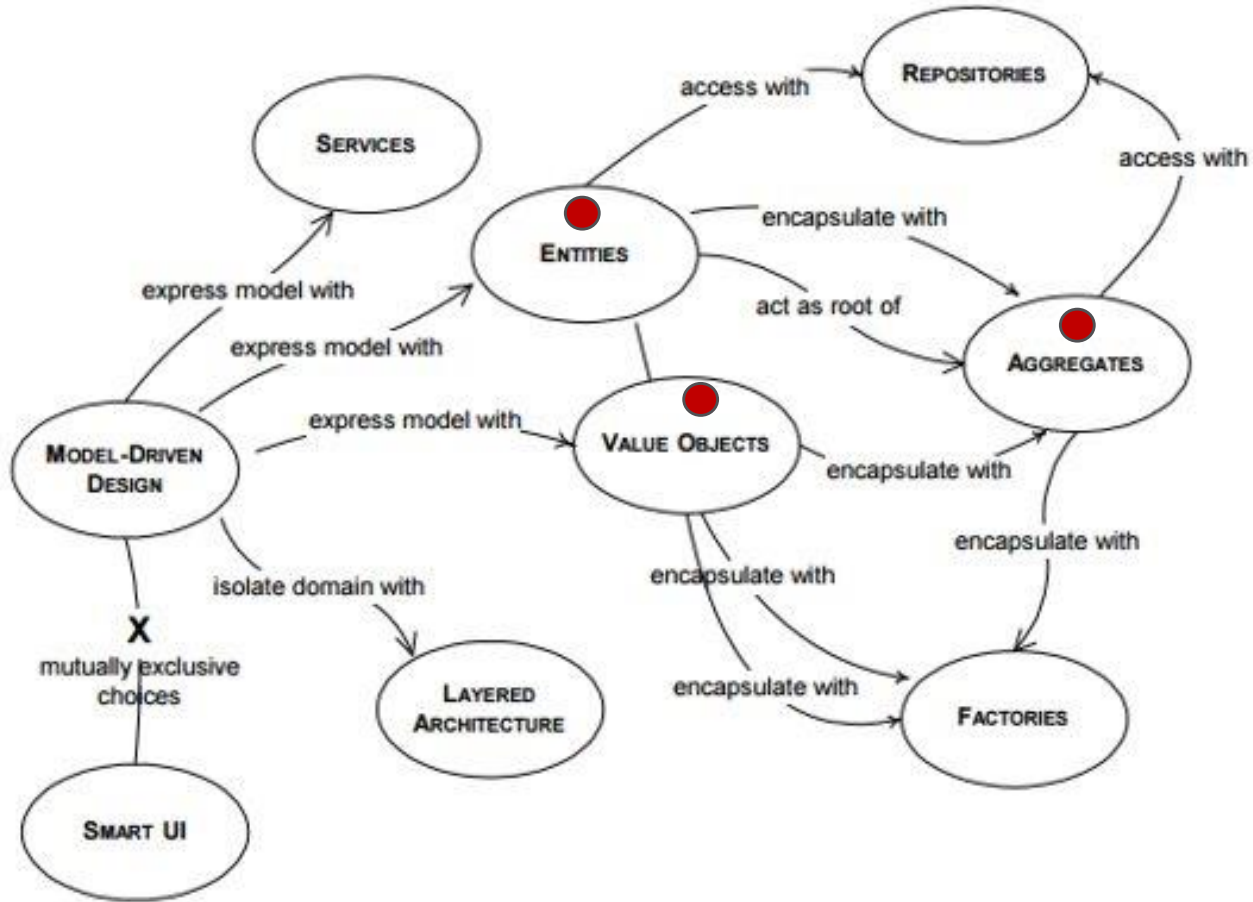
Présentation rapide du DDD



Le DDD Stratégique



Le DDD Tactique



Présentation de l'exemple

- Gestion de tournoi de tennis
 - Un joueur à un nom
 - Lors d'un match, deux joueurs s'affrontent
 - Un match commence, puis est en cours et s'arrête
 - Il y a plusieurs joueurs dans un tournoi
 - Un tournoi peut être ouvert, fermé, puis démarré et fini
 - Un joueur ne peut participer qu'à un match à la fois



Value-Object

Il est immutable

Il n'a pas de cycle de vie

Il est défini par ses
valeurs

Valeurs égales

Pas de Setter

Peut être échangé

Par exemple : une adresse, un point GPS...

Lequel est le value object?

- Le joueur
- Ou
- Le match



Exemple à travers le joueur



<https://github.com/xblanc33/championship/blob/master/java/src/main/java/championship/model/Player.java>



Analyse du code

Value-Object

- Un constructeur (avec un check)
- Une méthode pour récupérer les attributs
- Pas de Setter
- Un equals
- Une méthode hashCode et toString

```
class Player {  
    private String nickname;  
  
    Player(String nickname) {  
        if (nickname == null) {  
            throw new IllegalArgumentException("nickname cannot be null");  
        }  
        this.nickname = nickname;  
    }  
  
    public String getNickname() {  
        return nickname;  
    }  
  
    @Override  
    public boolean equals(Object other) {  
        if (!(other instanceof Player)) {  
            return false;  
        } else {  
            Player otherPlayer = (Player) other;  
            boolean equals = nickname.compareTo(otherPlayer.nickname) == 0;  
            return equals;  
        }  
    }  
  
    @Override  
    public int hashCode() {  
        return Objects.hash(getNickname());  
    }  
  
    @Override  
    public String toString() {  
        String res = "Player nickname=" + nickname;  
        return res;  
    }  
}
```

Project Lombok

Simplifications
de méthode

Peut être utilisé dans
les méthodes hashCode,
equals et toString

Comment ça marche?

Entity

OBJET "CLASSIQUE"

A UN CYCLE DE VIE

DÉFINI PAR SON ID

PEUT "AVOIR" DES VALUE-OBJECT

PEUT ÊTRE CONNECTÉ AUX AUTRES ENTITÉES

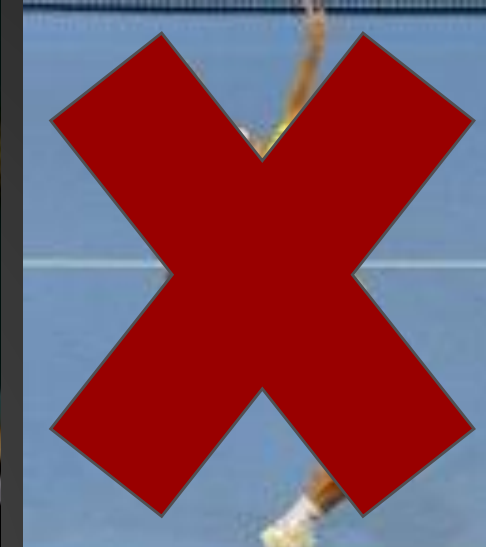
SETTER "D'ENTREPRISE" (PAS DE SETXXX)

ENCAPSULATION (ATTENTION AU GETTER)

RESPONSABLE

Lequel est l'entity?

- Le joueur
- Ou
- Le match



Exemple avec le match



<https://github.com/xblanc33/championship/blob/master/java/src/main/java/championship/model/Match.java> (modifié)



Analyse du code

Résumé Entity



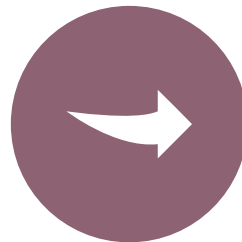
Un attribut id métier



Des "getter" pour
les attributs



Des "setters"



Des méthodes adaptées au
cycle de vie

Agregate

OBJET COMPOSITE

A UN CYCLE DE VIE

PEUT CONTENIR DES VALUES
OBJECT ET DES ENTITY

CONTROLE LE CYCLE DE VIE
DES OBJECTS QU'IL CONTIENT

Agregate



<https://github.com/xblanc33/championship/blob/master/java/src/main/java/championship/model/Championship.java>



Analyse du code

Agregate

Composite :
Que contient t-il?

Etat: lui même et sa
composition

Maitrise sur les
object qu'il contient

Exemple

- Gestion de tournoi de tennis
- Un joueur à un nom
- Lors d'un match, deux joueurs s'affrontent
- Un match commence puis est en cours et s'arrête
- Il y a plusieurs joueurs dans un tournoi
- Un tournoi peut être ouvert, fermé, puis démarré et fini
- Un joueur ne peut participer qu'à un match à la fois



Inconvénients du DDD

- Difficulté de convaincre les acteurs métier
- Difficulté d'apprentissage de l'ubiquitous language et du domaine
- Difficulté de représenter le Domain model
- Approche non adaptée aux projets où l'application est focalisée sur l'affichage et l'utilisation des données en ayant peu de logique métier.

Conclusion

Adaptable chaque métier

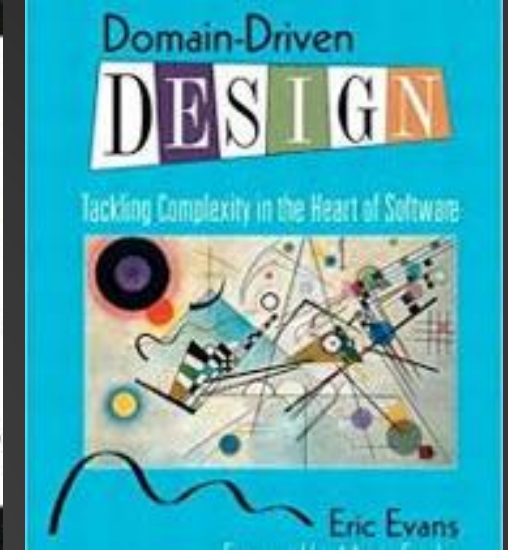
Stratégique et tactique

De plus en plus utilisé

Revoit totalement les règles de codage actuelles

Annexes

- Livres :
 - Tête la première: Design Patterns: Eric et Elisabeth Freeman
 - Le DDD Vite fait: Abel Avram et Floyd Marinescu inspiré de :
 - Domain-Driven Design: Tackling Complexity in the Heart of Software de Eric Evans
- Présentation :
 - Présentation de Xavier Blanc à la Bordeau JUG 2019 et son code disponible sur son git



Un petit quiz

<https://unil.im/quizzDDD>