

Redonner du sens aux
objets

C'est quoi le problème ?

- attributs
- constructeur
- getter, setter
- `equals()`, `toString()`

⇒ ce sont des coquilles vides

Et donc ?

- on éparpille les traitement dans le code
- on génère du code spaghetti
- on a des effets de bord imprévisible
- c'est hyper dur à faire évoluer

Comment on fait alors ?

- on remet les objets au centre du design

Domain Driven Design

Une méthode centrée sur un langage commun entre :

- les humains (développeurs, resp. métier, utilisateurs, ...)
- le code

⇒ et ça passe par les objets

- les *Value Objects*
- les *Entities*

Value objects

- une capsule
- donne du sens à une valeur primitive

Exemple: Email

```
class RegisterHandler {  
  
    public void register(String email, String password, String username) {  
        // impossible de faire confiance à ces valeurs  
        // il faut penser à les valider manuellement  
        if(  
            checkNotNull(email)  
            && checkNotEmpty(email)  
            && checkFormatValid(email)  
        ) {  
            // traitement  
        }  
    }  
}
```

Example: Email (suite)

```
class Email {  
    private final String value;  
  
    public Email(String email) {  
        this.value = email;  
    }  
  
    public String getValue() {  
        return value;  
    }  
}
```

Exemple: Email (avec validation)

```
class Email {  
    private final String value;  
  
    public Email(String email) {  
        checkNotNull(email).orThrowError("Email cannot be null");  
        checkNotEmpty(email).orThrowError("Email is mandatory");  
        checkFormatIsValid(email).orThrowError("Email is incorrect");  
        this.value = email;  
    }  
  
    public String getValue() {  
        return value;  
    }  
}
```

Exemple: Email (suite)

```
class RegisterHandler {  
  
    public void register(Email email, Password password, Username username) {  
        // plus besoin de protéger le code : les valeurs sont valides  
    }  
  
}
```

Exemple: Email (suite)

```
class UserController {  
  
    postRegister(RegisterFormData data) {  
        // avant  
        this.registerHandler.register(data.email(), data.username(), data.password)  
        // ✗ oups ! les paramètres username et password sont inversés  
  
        // après  
        this.registerHandler.register( // impossible de se tromper ici  
            new Email(email),  
            new Password(password),  
            new Username(username)  
        );  
    }  
}
```

Entities

- un objet représentant un concept métier
- utilise des *Value Objects*

Exemple : RecetteController (avant)

```
public class RecetteController {  
  
    public Recette getRecette(String idRecette, OptionsRecette options) {  
        Recette fetched = RecetteUtils.fetchRecetteById(idRecette);  
        return new Recette(  
            fetched.getId(),  
            fetched.getTitre(),  
            fetched.getIngredients().stream().map(ingredient -> new Ingredient(  
                ingredient.getNom(),  
                ingredient.getQuantite() * options.getNbPersonnes(),  
                ingredient.getUnite()  
            ))  
            .collect(Collectors.toList()),  
            fetched.getEtapes().stream().map(etape -> new Etape(  
                etape.getDescription(),  
                etape.getDuree() * (options.getNbPersonnes() / 2.0),  
                etape.getUnite()  
            ))  
            .collect(Collectors.toList())  
        );  
    }  
}
```

Exemple : Recette

```
class Recette {  
    public Recette(  
        private String id,  
        private String titre,  
        private List<Ingredient> ingredients,  
        private List<Etape> etapes  
    ) { /* ... */ }  
  
    adapteNbPersonnes(int nbPersonnes) {  
        return new Recette(  
            id,  
            titre,  
            ingredients.stream()  
                .map(ingredient -> ingredient.adapteNbPersonnes(nbPersonnes))  
                .toList(),  
            etapes.stream()  
                .map(etape -> etape.adapteNbPersonnes(nbPersonnes))  
                .toList()  
        );  
    }  
}
```

Exemple : RecetteController (après)

```
class RecetteController {  
  
    public Recette getRecette(String idRecette, OptionsRecette options) {  
        var recette = fetchRecetteById(id);  
        return recette.adapteNbPersonne(nbPersonnes);  
    }  
  
}
```

Récapitulatif

Value Objects

- On donne du sens aux éléments de base
- On sécurise le code

Entities

- On centralise les traitements dans l'objet

Merci de votre attention

