Redonner du sens aux objets

Le constat

Souvent une classe se résume à :

- des attributs
- un constructeur
- des getter & setter
- `equals()`, `toString()`

⇒ ce sont des coquilles vides

Et donc ?

- les traitements se font à l’extérieur de la classe
  - ie.: les classes "XxxService"

⇒ les traitements sont éparpillé dans la base de code

⇒ on génère du code spaghetti

⇒ on a des effets de bord imprévisible

😱 C’est l’enfer à maintenir

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 le code**

- 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
        }
    }
}
```

Exemple : Créer un value object

```
class Email {
    private final String value;

    public Email(String email) {
        this.value = email;
    }

    public String getValue() {
        return email;
    }
}
```

Exemple : Ajout de la 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 email;
    }
}
```

Exemple : Utilisation du value object

```
class RegisterHandler {

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

}
```

Exemple : Cas d'utilisation

```
class UserController {

    public void postRegister(RegisterFormData data) {
        // avant
        this.registerHandler.register(data.email(), data.username(), data.passwor
        // ❌ oups ! les paramètres username et pasword sont inversés

        // après
        this.registerHandler.register( // impossible de se tromper ici
            new Email(data.email()),
            new Password(data.password()),
            new Username(data.username())
        );
    }
}
```

Entities

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

Exemple : RecetteService (avant)

```
public class RecetteService {
    public Recette getRecette(String idRecette, OptionsRecette options) {
        Recette recette = RecetteUtils.fetchRecetteById(idRecette);

        for (Ingredient ingredient : recette.getIngredients()) {
            ingredient.setQuantite(ingredient.getQuantite() * options.getNbPersor
        }

        for (Etape etape : recette.getEtapes()) {
            etape.setDuree(etape.getDuree() * (options.getNbPersonnes() / 2.0));
        }

        return recette;
    }
}
```

Exemple : Recette Entity

```
public class Recette {

    private final String id;
    private final String titre;
    private final List<Ingredient> ingredients;
    private final List<Etape> etapes;
    /* ... */
    public Recette 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 : RecetteService (après)

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

Récapitulatif

Avec les Value Objects

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

Avec les Entities

- On centralise les traitements dans l'objet

Merci de votre attention