Diagramme de séquence, Injection de dépendance, Builder

Exercice 1 - Diagramme de séquence

En reprenant les classes du premier td td3convert.zip (`td3convert.zip`) sur les filtres d'images.

1 À partir du code écrit précédemment modéliser le scénario d'applications d'un ensemble de filtre avec une image en utilisant un diagramme de séquence avec les classes que vous avez écrites.
Sur le paquetage courant, créer un "Diagramme de séquence", placer l'acteur utilisateur à gauche et indiquer l'ensemble des méthodes qui sont appelées.

Exercice 2 - Injection de dépendance

1 Dans un souci d'internationalisation, on souhaite que le fichier des filtres permette de définir un filtre soit par son nom en anglais ou en français (`rotation` et `rotate`, par exemple).
L'idée consiste à permettre à chaque créateur de filtre de s'enregistrer plusieurs fois avec des noms de filtre différents dans la table d'association.
Ajouter une méthode indiquant pour chaque créateur de filtre l'ensemble des noms pour lesquels il veut être enregistré.

2 Modifier votre code pour qu'au lieu d'utiliser une méthode de description on utilise une annotation `FilterNames` permettant de définir l'ensemble des noms d'un filtre.
Changer votre code en conséquence.

3 De façon plus général pour résoudre ce genre de problème, on utilise le design-pattern Injection de dépendance (appelé aussi Inversion de contrôle). C'est à dire, au lieu d'enregistrer le créateur de filtre pour un ensemble de nom dans un code externe, on demande au créateur de filtre de s'enregistrer lui-même en lui passant en paramètre une table d'association ou plus exactement une abstraction de table associative pour éviter un couplage trop fort.
Implanter ce design-pattern en permettant que par défaut l'utilisation de l'annotation `FilterNames`.

Exercice 3 - Requète SQL

On souhaite créer une classe `Query` permettant de créer des requêtes SQL. Nous nous limiterons à des requêtes `SELECT ... FROM ... WHERE ... AND ...` (avec autant de AND que l'on veut).
Pour créer la requête, on souhaite permettre à l'utilisateur de spécifier uniquement des couples nom de table/champs soit en tant que clause `SELECT` soit en tant que clause `WHERE`. Les valeurs du `FROM` seront calculées automatiquement.

1 Implanter la classe `Query` en utilisant le design-pattern builder :

```
Query query=new Query();
query.select("user.id");
```

```
query.select("user.name");
String sql=query.toSQL();
```

le code précédent doit générer la requête :

```
SELECT user.id,user.name FROM user;
```

2 Faire en sorte que l'on puisse chaîner les requêtes :

```
new Query().select("user.id").select("user.name").toSQL();
```

3 On souhaite de plus gérer les clauses WHERE sachant que l'on peut comparer la valeur de deux champs avec '='.
Attention, la solution que vous proposerez devra éviter d'avoir à écrire un parseur d'expressions.
Mdifier votre code pour que l'on puisse créer une requête de ce type :

```
SELECT id,name,street FROM user, address WHERE
user.id = address.user_id;
```

Attention à la gestion des constantes `... WHERE user.id=3`

4 Ajouter la gestion des '<>', '<', '>'.

5 Créer une annotation `@Table` qui permet d'assigner un nom de table à une classe. Faire en sorte que l'annotation soit accessible par reflexion lors de l'exécution, et se transmette aux sous-types.
Ajouter une méthode `select(Class<?> type,String field)` qui va chercher le nom de la table en utilisant la classe `type`.