TD3 : Factory, Singleton, Builder

Exercice 1 – Factory

Le but de cet exercice est de créer un petit programme permettant d'appliquer une série de filtres à une série d'images.
Commencez par télécharger le fichier td2convert.zip (http://igm.univ-mlv.fr/ens/IR/IR2/2008-2009/POO/td2convert.zip). Puis importez-le dans le répertoire source de votre projet courant dans Eclipse.
Pour cela, importez l'archive dans Eclipse. avec clic droit sur votre répertoire contenant les sources sélectionnez "import" puis "archive file". Indiquez alors le fichier téléchargé.
Au besoin, faites un clic droit sur le paquetage importé, puis sélectionnez "refactor" pour donner un nom qui correspond à l'arborescence voulue.
Voici un exemple de fichier (`filter.txt`) décrivant une succession de filtres à appliquer sur une image :

```
gray
rotate
blur
```

Avec la ligne de commande suivante (au nom de package près) :

```
java Convert filter.txt image.gif
```

Le programme va appliquer l'ensemble des filtres décrit dans le fichier `filter.txt` à l'image `image.gif` (récupérez celle que vous voulez) et ressortir le résultat sous forme d'une image PNG nommée `image.gif.png`.

1. Pourquoi les constructeurs de `Images` et de `ImageOps` sont déclarés `private` ?
2. Proposez une solution de conception permettant, sans changer les classes existantes, de pouvoir appliquer indifféremment n'importe quel filtre à une image.
   Codez la solution proposée.
3. Faites le schéma UML de ce que vous venez de coder
4. Dans le but d'associer un filtre à une chaîne de caractères, écrivez une méthode `createFilter(String name)` qui renvoie un filtre en fonction de son nom.
   Dans quelle classe doit-on mettre cette méthode ? Cette méthode doit-elle ou non être statique ?
   Comment s'appelle ce design pattern ?
5. Remarquez que les objets filtres sont indépendants de l'image qu'il manipule. Il est donc possible de renvoyer plusieurs fois le même filtre pour une même opération.
   Changez votre code en correspondance.
6. Comment faire pour eviter les `if .. else` dans le code de la méthode `createFilter(String name)` ?
   Changez votre code en conséquence.
7. Écrivez le code qui prend en paramètre un fichier et ressort sous forme d'une liste l'ensemble des filtres à appliquer; vous utiliserez pour cela la classe `java.util.Scanner`.
8. On souhaite maintenant pouvoir choisir dans le code précédent la façon dont les filtres sont implantés (builtin JDK ou librairie graphique opencv ou ...). Que doit-on faire ?
   Modifiez votre code en conséquence.
9. Écrivez la classe `Convert` et testez la.

Exercice 2 – Un objet ... un seul !

On veut avoir un pool de connexions à une DB.

```
Class DBConnexionPool {
      private List<DBConnexion> connexions = new ArrayList<DBConnexion>();

      public synchronized DBConnexion getConnexion() {
            for ( DBConnexion c : connexions) {
                  if ( ! c.isBusy) {
                        c.setBusy();
                        return c;
                  }
            }
            DBConnexion c = new DBConnexion();
            c.setBusy();
            connexions.add(c);
            return c ;
      }
      public synchronized void releaseConnexion(DBConnexion c) {
            c.setFree();
      }
}
```

Quel est le problème si chaque utilisateur de cette classe crée une instance ?
Comment faire de cette classe un *Singleton* ?

La solution intuitive est-elle thread-safe ?
donnez 3 solutions différentes pour la rendre thread-safe ?
Laquelle ou lesquelles préconisez-vous ?

Pourquoi ne pas juste rendre la classe totalement statique ?

Exercice 3 - Requête 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. Implantez la classe `SqlQueryBuilder` en utilisant le design-pattern builder :

   ```
   SqlQueryBuilder query=new SqlQueryBuilder();
   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. Faites en sorte que l'on puisse chaîner les requêtes :

```
new SqlQueryBuilder().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 '='.
   La solution proposée ne devra pas utiliser de parseur d'expressions.
   Modifiez 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. Ajoutez la gestion des '<>', '<', '>'.

Exercice 4 - Filtre avec argument

En reprenant l'exercice sur les filtres, on souhaite maintenant permettre que les filtres puissent prendre des paramètres :

```
gray
rotate 3.14
blur
```

Attention, tous les changements effectués doivent permettre que les anciens fichiers de filtrage continuent à fonctionner.

1. Quelles parties du code précédent doit-on changer pour prendre en compte le changement de la spécification du fichier des filtres ?
   Attention, petit piège, il y a plusieurs parties qui changent !
2. Quel design pattern doit-on utiliser pour que le parseur du fichier des filtres délègue à chaque créateur de filtre le soin de lire les arguments du filtre correspondant?
3. Modélisez avec un diagramme de séquence les échange entre les différents objets.
   Implantez ce design pattern.