TD4 : Factory, Decorator, Composite

Exercice 1 – Decorator et Composite

Pour illustrer les possibilités de ces design patterns avec les flux, on veut :

- écrire des classes `UpperCaseWriter et LowerCaseWriter` qui permettront de forcer l'écriture en majuscule ou en minuscule
  - faites le diagramme UML du Design Pattern que vous venez d'utiliser
- écrire une classe `CompositeWriter` qui permettra (entre autres) d'écrire de manière simultanée dans un fichier avec le texte en minuscule et dans un autre fichier avec le texte en majuscule.

Exemple du code client que l'on veut pouvoir écrire :

```
CompositeWriter cw = new CompositeWriter();
cw.add(new UpperCaseWriter(new FileWriter(new File("test-u"))));
cw.add(new LowerCaseWriter(new FileWriter(new File("test-l"))));
Reader isrf = new UpperCaseReader(
             new BufferedReader(
             new InputStreamReader(
             new FileInputStream(
             new File("src\\fr\\umlv\\poo\\dp\\composite\\books.xml")))));
int c;
while ((c = isrf.read()) > 0) {
       cw.write(c);
}
cw.close();
```

Une fois que votre code fonctionne, que se passe-t-il si l'écriture dans le fichier est faite avec une des autres méthodes write(), par exemple write(char[] cbuf) ?

Exemple avec lecture et écriture par 100 chars

```
char[] t = new char[100];
CompositeWriter cw2 = new CompositeWriter();
cw2.add(new UpperCaseWriter(new FileWriter(new File("test-u2"))));
cw2.add(new LowerCaseWriter(new FileWriter(new File("test-l2"))));
Reader isrf2 = new BufferedReader(
             new InputStreamReader(
             new FileInputStream(
             new File("src\\fr\\umlv\\poo\\dp\\composite\\books.xml"))));
while (isrf2.read(t) > 0) {
       cw2.write(t);
}
cw2.close();
```

Que faut-il faire pour résoudre le problème ?

Exercice 2 – 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 td-convert.zip (http://igm.univ-mlv.fr/ens/IR/IR2/2012-2013/ProgOO/td-convert.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` (ou un `switch`) 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 ...). Imaginez qu'un concours soit organisé entre UMLV et Berkeley pour savoir quelle université codera les filtres les plus performants. Que doit-on faire ?
   Modifiez votre code en conséquence.
9. Écrivez la classe `Convert` et testez la.

Exercice 3 - Filtre avec argument (à faire chez vous!)

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. Faites en sorte que le parseur du fichier des filtres délègue à chaque créateur de filtre le soin de lire les arguments du filtre correspondant. Comment ?
3. Modélisez avec un diagramme de séquence les échange entre les différents objets.
4. Implantez votre solution.