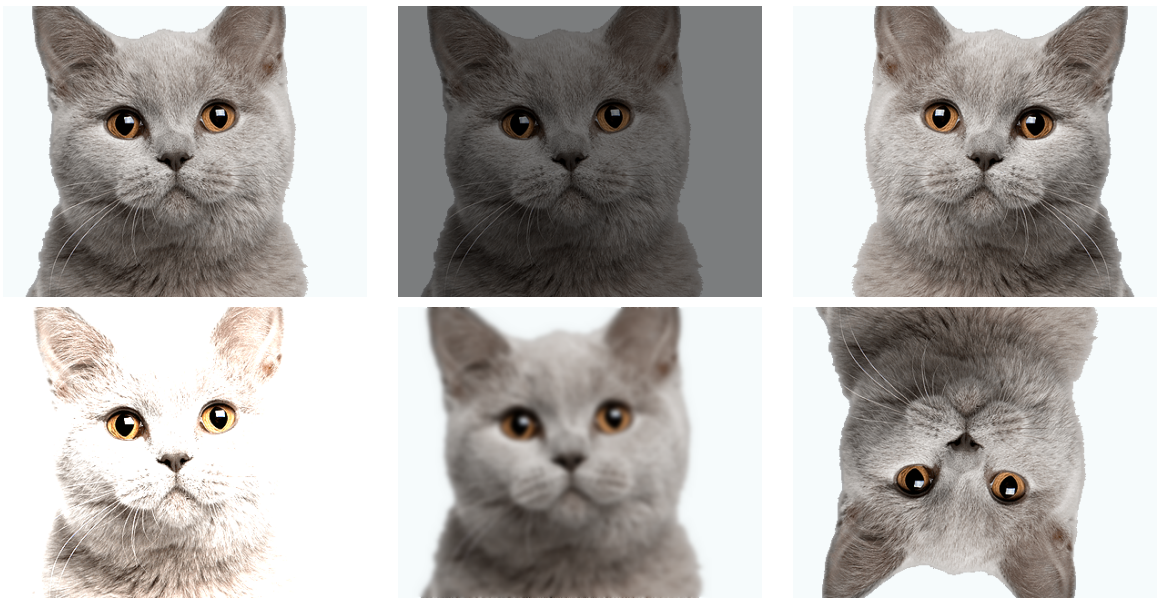

Table 1: L'image originale avec les cinq nouvelles versions : assombri, éclairci, flou, renversée X et Y.

TP 4: Images

1 Traitement d'images

Pour cet exercice vous devez écrire un programme qui, étant donné un fichier qui contient une image en format PPM, construit un autre fichier PPM qui va contenir une version modifiée de l'image originale. Vous devez implementer les operations suivantes :

1. Assombrir. L'image devient plus sombre. Vous pouvez effectuer cette operation en divisant toutes les valeurs par 2.
2. Éclaircir. L'inverse de l'operation précedente. Attention! Aucune valeur ne peut dépasser la valeur maximum declaré au début du fichier.
3. Flou. Cette opération donne une version flou de l'image en remplaçant chaque pixel par la moyenne de son voisinage.
4. Renverser X et Renverser Y. Cette opération donne une version symmetrique de l'image (miroir).

La Table 1 montre un exemple.

1.1 Fonctionnement

Votre programme doit prendre trois paramètres : l'opération à effectuer, le fichier qui contient l'image, et le fichier où on va mettre sa nouvelle version. Exemple: `java Image darken cat.ppm cat-dark.ppm`

doit prendre l'image contenue dans le fichier `cat.ppm` et mettre une version assombrie de cette image dans le fichier `cat-dark.ppm`.

1.2 Les Fichiers PPM

Un fichier PPM est un fichier graphique, c'est-à-dire qu'un tel fichier contient les informations necessaires pour décrire une image. Il y a plusieurs types de fichiers PPM mais pour cet exercice on va utiliser seulement les fichiers de type P3, qui sont des fichiers ASCII (plain text) et décrivent des images en couleur.

Un fichier PPM de ce type commence avec le string "P3". Le reste du fichier est composé par une séquence des entiers, separés par espaces ou nouvelles lignes. Les deux premiers nombres sont la largeur et la hauter de l'image. Le nombre qui suit, qui est d'habitude 255, est la valeur maximum contenu dans le fichier. Après ces trois nombres, l'image est codée ligne par ligne en partant du haut, chaque ligne étant codée de gauche à droite. Chaque pixel est codée par trois nombres, les valeurs rouge, verte et bleue (red, green, blue) de la couleur du pixel.

(Normalement, les fichiers PPM peuvent aussi contenir des commentaires commençant par #, mais pour cet exercice vous pouvez supposer que les commentaires ne sont pas permis.)

Vous pouvez trouver une réference pour ce type de fichier, par exemple ici.

1.3 Lire un fichier

Pour lire des informations qui sont stockées dans un fichier PPM de format texte vous pouvez utiliser les classes `File` et `Scanner`. On a déjà vu quelques exemple de l'utilisation de `Scanner`. L'idée ici est de construire un objet de type `Scanner` dont la source ne sera pas le clavier (`System.in`), mais un fichier. Un objet de la classe `File` représente un fichier. Par exemple vous pouvez utiliser le code suivant :

```
1  File file = new File(filename);
2  Scanner sc = null;
3  try{
4          sc = new Scanner(file);
5                  /* ... sc.nextInt(); ou sc.next(); */
6  }catch(IOException e){
7                  System.out.println("File I/O error!");
8                  e.printStackTrace();
9  }finally{
10                 if(sc!=null) sc.close();
11 }
```

Attn : le block `finally` guarantit que le fichier sera fermé. Le block `try..catch` est necessaire, parce que `IOException` est une exception qui peut être lancée par `Scanner` (est `IOException` est une exception "checked").

Attn : n'oubliez pas de faire `import` pour les classes que vous utilisez !

1.4 Écrire un fichier

Pour écrire un fichier vous pouvez utiliser la classe `PrintWriter`. Une fois que vous avez construit un objet de cette classe avec le constructeur qui prend comme paramètre le nom du fichier, vous pouvez utiliser la méthode `println`, comme on fait avec `System.out`. Exemple :

```
PrintWriter writer = null;
try{
        writer = new PrintWriter(filename);
        //writer.println("blabla");
}catch(IOException e){
        System.out.println("File I/O error (write)!");
        e.printStackTrace();
}finally{
        if(writer!=null) writer.close();
}
```

1.5 Représenter une image

Pour représenter une image vous devez d'abord écrire une classe `Pixel` qui représente un élément de l'image, c'est-à-dire les valeurs RGB de l'élément. Puis, la classe `Image` décrit les objets qui contiennent une matrice de Pixels, alors que quelques informations supplementaires (si necessaire). Compléter le code suivant :

```
class Pixel{
        int red;
        int green;
        int blue;
/* .. */
}

class Image{
        Pixel [][] pixels;
        int width, height,max;
        public Image(String filename){ /*Reads image from PPM file*/ }
        public Image darken(){ /*..*/ }
        public Image lighten(){ /*..*/  }
        public Image blur(){ /*..*/ }
        public Image reverseX(){ /*..*/ }
        public Image reverseY(){ /*..*/ }
        public void write(String filename){ /*..*/ }
        /*..*/
}
```