Aix*Marseille université
Initiative d'excellence
Année 2020-21

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 4 (Prog2)
Code UE : SIN3U02

TP4 : Manipulation d'images

1 Manipulation d'images

On s'intéresse dans ce TP à la représentation et à la manipulation d'images. Ces images sont constituées de pixels. Chaque pixel est caractérisé par des coordonnées et une couleur. Dans notre cas, on ne s'intéressera qu'à des images en noir et blanc et la couleur de chaque pixel sera en fait un niveau de gris.

1.1 Niveaux de gris

Le niveau de gris d'un pixel est représenté par un objet d'une classe qui implémente l'interface `GrayColor` :

```
public interface GrayColor extends Comparable<GrayColor> {
    double getLuminosity();
    Color getColor();
}
```

La luminosité est un nombre décimal compris entre 0 (noir) et 1 (blanc) qui représente les nuances de gris. Une classe implémentant l'interface `GrayColor` doit donc implémenter les trois méthodes suivantes :

- `double getLuminosity()` : renvoyant le niveau de gris de la couleur compris entre 0 et 1.
- `Color getColor()` : renvoyant une couleur pour l'affichage.
- `compareTo(GrayColor o)` : compare la luminosité de deux couleurs, les couleurs sombres étant considérées plus petite pour l'ordre de compareTo. On pourra utiliser la méthode `Double.compare` pour comparer les deux `double`.

Une représentation classique pour les niveaux de gris considérés consiste à utiliser un entier sur un octet (*byte* en anglais), donc une valeur entre 0 (noir) et 255 (blanc). Cette représentation est implémentée par la classe `ByteGrayColor`.

1.1.1 Tâche 1 : classe `ByteGrayColor`

Compléter la classe `ByteGrayColor` implémentant l'interface `GrayColor`. La méthode `Color getColor()` est déjà implémentée. En plus de compléter les méthodes, il vous faudra compléter les trois constructeurs suivants :

- `ByteGrayColor()` : construit une couleur avec un niveau de gris égal à `MINIMUM_GRAY_VALUE`.
- `ByteGrayColor(int grayLevel)` : construit une couleur avec un niveau de gris égal à l'argument.
- `ByteGrayColor(double luminosity)` : construit une couleur à partir de la luminosité désirée comprise entre 0 et 1.

On a ajouté aussi deux constantes publiques pour les couleurs noir et blanc.

Lorsque vous accomplissez un `TODO`, dans le code qui vous est fourni, supprimez le commentaire correspondant. N'oubliez pas de faire régulièrement des *commits*.

1.2 Définition des images

Les images en niveau de gris correspondent à l'interface suivante :

```
public interface GrayImage extends Image {
    void setPixel(GrayColor gray, int x, int y);
    GrayColor getPixelGrayColor(int x, int y);
}
```

L'interface `Image` étant définie par :

```
public interface Image {
    Color getPixelColor(int x, int y);
    int getWidth();
    int getHeight();
}
```

Une classe implémentant l'interface `GrayImage` doit donc implémenter les cinq méthodes suivantes :

— `GrayColor getPixelGrayColor(int x, int y)` : renvoie la `GrayColor` du pixel (x, y).
— `Color getPixelColor(int x, int y)` : renvoie la couleur du pixel (x, y) pour l'affichage.
— `void setPixel(GrayColor gray, int x, int y)` : change la couleur du pixel (x, y).
— `int getWidth()` : renvoie la largeur de l'image.
— `int getHeight()` : renvoie la hauteur de l'image.

La coordonnée `x` d'un pixel représente sa position horizontale, et sa coordonnée `y`, sa position verticale. Le pixel de coordonnées $(0, 0)$ est le pixel situé en haut à gauche de l'image. L'axe des `x` est donc orienté vers la droite et celui des `y` vers le bas.

1.2.1 Tâche 2 : classe `MatrixGrayImage`

Compléter la classe `MatrixGrayImage` implémentant l'interface `GrayImage`. En plus de compléter les méthodes, il vous faudra compléter le constructeur `MatrixGrayImage(int width, int height)` qui initialise une image de taille `width` × `height`. La matrice `pixels` stocke les couleurs des pixels de l'image : la case en ligne `x` et colonne `y` (`pixel[x][y]`) contient donc la couleur du pixel (x, y). On initialisera une image en mettant tous ses pixels à blanc.

La classe `MatrixGrayImage` possède la méthode `createImageFromPGMFile(String fileName)` qui permet de charger une image au format `pgm`. Un exemple d'image vous est donné dans le fichier `luminy.pgm`. Si vous chargez cette image avec la méthode `createImageFromPGMFile`, vous devriez obtenir l'affichage suivant :

2 Transformations d'images

On souhaite faire des manipulations simples d'images. Pour cela, nous avons défini l'interface `Transform` suivante :

```
public interface Transform {
    void applyTo(GrayImage image);
}
```

Un appel à la méthode `applyTo` d'une classe qui implémente l'interface `Transform` devra modifier l'image correspondant à la transformation.

Vous allez définir trois classes implémentant cette interface. La classe `Invert`, qui va inverser le niveau de gris des pixels d'une image, la classe `DecreaseGrayLevels`, qui va réduire le nombre de valeurs de gris différentes et la classe `Pixelate`, qui va réduire la définition d'une image.

2.1 Inversion des niveaux de gris

La première transformation consiste à modifier chaque pixel de l'image de sorte que le nouveau niveau de gris de chaque pixel soit égal au niveau de gris maximum auquel on soustrait l'ancien niveau de gris.

2.1.1 Tâche 3 : classe `Invert`

Avant d'écrire la classe `Invert` qui va nous permettre de réaliser la transformation d'image, on va procéder aux changements suivants :

— Ajouter à l'interface `GrayColor` la méthode `GrayColor invert()`.

— Implémenter `invert()` dans `ByteGrayColor`.

Définissez, si vous ne l'avez pas encore fait, l'interface `Transform` puis créer la classe `Invert` implémentant l'interface `Transform` et compléter cette classe. La méthode `applyTo(GrayImage image)` de la classe `Invert` devra transformer l'image passée en argument, en remplaçant la couleur de chaque pixel par son inverse calculé par la méthode `invert()`.

Une fois la classe `Invert` implémentée, modifiez la méthode `main` de la classe `Main` de façon à appliquer la transformation à la variable `image` après le chargement de l'image (appel à `createImageFromPGMFile`) et avant son affichage (appel à la méthode `display`). Vous devriez obtenir l'affichage suivant :

2.2 Diminution du nombre de niveaux de gris

On cherche maintenant à modifier une image en diminuant le nombre de niveaux de gris. Ce nombre de niveaux de gris (un entier) sera l'attribut `nbGrayLevels` de la classe `DecreaseGrayLevels` implémentant

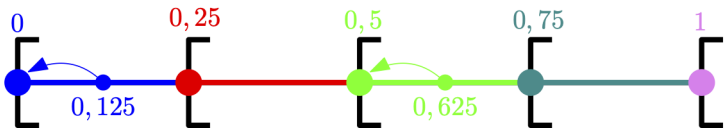

FIGURE 1 – Passage à 5 niveau de gris

`Transform`.

Pour diminuer le nombre de niveau de gris, il faut décomposer l'intervalle $[0, 1[$ en `nbGrayLevels - 1` sous-intervalles de taille `1/(nbGrayLevels - 1)`. Par exemple, pour 5 niveaux de gris, on découpe l'intervalle $[0, 1[$ en quatre partie égales : $[0, \frac{1}{4}[$, $[\frac{1}{4}, \frac{2}{4}[$, $[\frac{2}{4}, \frac{3}{4}[$ et $[\frac{3}{4}, 1[$. Une couleur ayant une luminosité dans un de ces intervalles est transformée en la couleur correspondant à la luminosité du début de l'intervalle. Par exemple, pour 5 niveaux de gris, la couleur ayant une luminosité de $0,125$ est transformé en la couleur de luminosité 0 et celle ayant une luminosité de $0,625$ est transformée en la couleur de luminosité $0,5$. La couleur de luminosité 1 reste de luminosité 1.

Pour réaliser cette transformation vous pourrez utiliser la fonction `Math.floor(double a)`

2.2.1 Tâche 4 : classe `DecreaseGrayLevels`

Créer la classe `DecreaseGrayLevels` implémentant l'interface `Transform` et la compléter.

Une fois la classe `DecreaseGreyLevels` implémentée, modifiez la méthode `main` de la classe `Main` de façon à appliquer la transformation à la variable `image` après le chargement de l'image (appel à `createImageFromPGMFile`) et avant son affichage (appel à la méthode `display`). Vous devriez obtenir l'affichage suivant :

2.3 Pixelisation

L'idée de cette transformation est de découper l'image en carrés d'une certaine taille. La figure ci-dessous illustre un découpage en carré de taille 10.

Pour chaque carré, on calcule le niveau de gris moyen de ses pixels puis on modifie tous les pixels du carrés pour qu'ils aient ce niveau de gris. La taille d'un côté d'un tel carré sera défini par l'attribut `newPixelSize` de la classe `Pixelate`.

2.3.1 Tâche 5 : classe `Pixelate`

Créer la classe `Pixelate` implémentant l'interface `Transform`.

Une fois la classe `Invert` implémentée, modifiez la méthode `main` de la classe `Main` de façon à appliquer la transformation à la variable `image` après le chargement de l'image (appel à `createImageFromPGMFile`) et avant son affichage (appel à la méthode `display`). L'attribut `newPixelSize` de l'instance de `Pixelate` devra être égal à `10`.

3 Tâches supplémentaires

3.1 Miroir

Créer une classe de transformation permettant de retourner l'image, soit verticalement soit horizontalement, soit les deux.

3.2 Images en couleurs

Rajouter des classes et extensions pour gérer les images en couleurs au format .ppm similaire au pmg sauf que les couleurs sont définies par trois entiers au format RGB

3.3 Menu

Rajouter un menu dans l'application permettant de contrôler les transformations et la lecture/écriture de fichier.