Aix*Marseille université
Initiative d'excellence
Année 2018-19

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 3 (math-info)
Code UE : SIN3U02

Manipulation d'images

1 Introduction

On s'intéresse à la représentation et la manipulation d'images. Ces images seront constituées de pixels caractérisés par un nombre représentant un niveau de gris.

1.1 Consignes

Comme pour le TP précédent et celui d'avant, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du premier TP.

Lien vers la classroom du TP : lien devoir github classroom

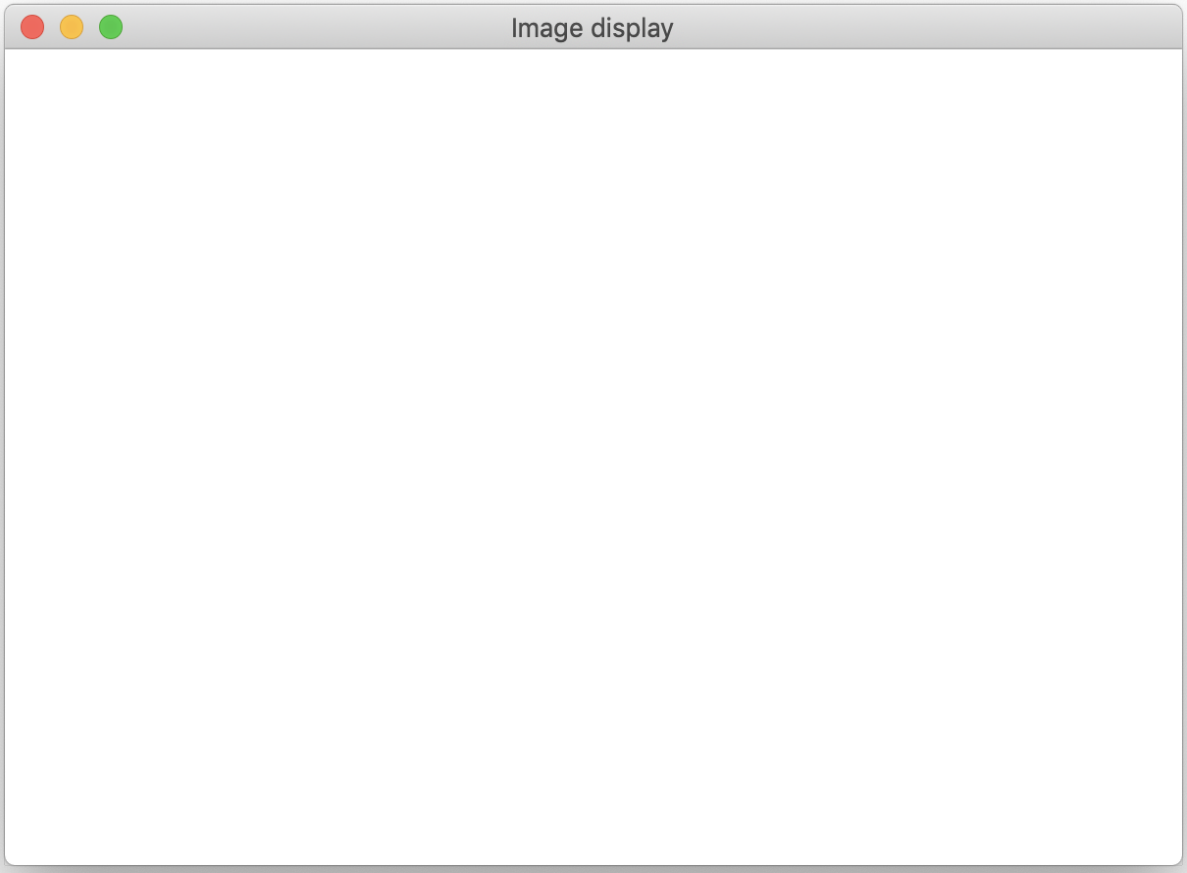
Après avoir cloner le dépot du TP, vérifier qu'IntelliJ a bien un JDK configuré. Pour cela, aller dans `File` -> `Project Structure`, onglet Projet. Il faut que le champ du `Project SDK` soit configuré. Si ce n'est pas le cas, modifiez-le en sélectionnant le répertoire d'installation du JDK sur votre ordinateur (sur les ordinateurs de l'université, c'est `/usr/local/jdk-13.0.2`). Réglez le `Project language level` au même niveau que le JDK.

Ce projet utilise Gradle. Gradle est un outil permettant d'automatiser les tâches pouvant être effectuées sur votre projet. Typiquement nous allons utiliser deux tâches : - compiler et lancer le projet, - exécuter les tests du projet. En particulier, Gradle est informé des librairies externes nécessaires au projet, et va donc les installer lui-même. Nous n'avons donc plus besoin de configurer IntelliJ et le projet, mais seulement Gradle.

La configuration de Gradle se fait avec le fichier `build.gradle` dans la racine du projet. Il est fourni déjà prêt (ou presque). Vous pouvez l'ouvrir et regarder ce qu'il y a dedans. Sans forcément tout comprendre, vous pouvez repérer plusieurs éléments indiquant les librairies utilisées par le projet.

Si vous utilisez une version de Java JDK différente de 13, il vous faut le signaler dans les champs `sourceCompatibility` et `targetCompatibility`, en remplaçant le 13 par le numéro de votre version.

Une fois le dépôt téléchargé, il vous faut donc absolument **importer le projet gradle** (soit en cliquant dans le popup s'ouvrant en bas à droite d'IntelliJ, soit en cliquant droit sur le fichier `build.gradle` du projet et en sélectionnant l'option adéquate), pour indiquer à IntelliJ que nous utilisons Gradle. Vous pouvez ensuite compiler et exécuter le programme en cliquant deux fois sur `nomDuProjet` -> `application` -> `run`, dans la sous-fenêtre Gradle d'IntelliJ (en haut à droite). Vous devriez obtenir l'affichage suivant.



Vous pouvez aussi exécuter les tests avec `nomDuProjet` -> `verification` -> `test`.

2 Définitions des couleurs et des images

2.1 Définition des couleurs

Les fonctionnalités d'une couleur représentant un niveau de gris sont décrites dans l'interface `GrayColor` :

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

La luminosité est un nombre flottant compris entre 0 (noir) et 1 (blanc) qui représente de façon continue les nuances de gris.

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

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)` : renvoie la comparaison des luminosités des 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`.

2.1.1 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.

Vous aurez besoin de quelques fonctions mathématiques de la classe Math pour les arrondis, à choisir parmi : - `long Math.round(double)` permet d'arrondir un double à l'entier le plus proche (3.7 devient 4), - `double Math.floor(double)` permet d'arrondir un double au double entier supérieur le plus proche (3.7 devient 4.0), - `double Math.ceil(double)` permet d'arrondir un double au double entier inférieur le plus proche (3.7 devient 3), - `int Math.toIntExact(long)` convertit un long en entier, en émettant une erreur si l'entier long n'est pas encodable sur 32 bits, - `Math.min` et `Math.max` permettent de calculer le minimum et le maximum de deux valeurs du même type numérique, - pour convertir un entier en double, vous pouvez utiliser un cast `(double) myInt`.

Ajouter aussi deux constantes publiques pour les couleurs noir et blanc.

Lorsque vous accomplissez un `TODO`, supprimer le commentaire correspondant. N'oubliez pas de faire régulièrement des *commits*.

2.2 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 cette interface 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.

Pour une image, x représente la coordonnée « horizontale » et y la coordonnée « verticale ». Le point de coordonnées $(0, 0)$ est le point 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.

2.2.1 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 un 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.

Vous devriez obtenir l'affichage suivant :

3 Transformations d'images

On souhaite faire des manipulations simples d'images. Pour cela, vous allez définir l'interface `Transform` suivante :

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

Un appel à la méthode `applyTo` devra modifier l'image suivant la transformation.

Vous allez donc définir des classes implémentant cette interface.

3.1 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.

3.1.1 Tâche 3 : classe `Invert`

Avant d'écrire la classe `Invert` qui va nous permettre de faire 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 remplaçant la couleur de chaque pixel par son inverse calculé par la méthode `invert()`.

Changer les instructions de la méthode `initialize` à l'intérieur de la classe `Display` de sorte que vous appliquiez la transformation `Invert` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`).

Vous devriez obtenir l'affichage suivant :

3.2 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 sera une propriété `nbGrayLevels` de la classe `DecreaseGrayLevels` implémentant `Transform`, qu'on supposera être un entier.

Pour diminuer le nombre de niveaux de gris, on va multiplier la luminosité (entre 0 et 1) par le nombre de niveaux de gris voulus moins 1 (on obtient un double entre 0 et `nbGrayLevels` - 1), et on arrondi à l'entier le plus proche.

3.2.1 Tâche 4 : classe `DecreaseGrayLevels`

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

Changer la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `DecreaseGrayLevels` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `nbGrayLevels` de l'instance de `DecreaseGrayLevels` devra être égal à `5`.

Vous devriez obtenir l'affichage suivant (approximativement, cela dépend de la façon dont vous avez fait les arrondis) :

3.3 Dessin des contours

Vous allez créer une classe `Outline` qui modifie une image par extraction de contours.

Il n'est pas très compliqué de réaliser une extraction de contour. En effet, une manière de procéder est la suivante : un pixel de l'image résultat est noir s'il appartient au contour de l'image initiale, c'est-à-dire s'il est très différent de l'un des deux pixels situés à sa droite ou en-dessous de lui dans l'image initiale. L'expression **Très différent** signifie que la valeur absolue (fonction `Math.abs(double a)` de la différence entre les luminosités de deux pixels est supérieure à un seuil fixé. Les autres pixels de l'image (qui ne sont pas identifiés comme appartenant à un contour) sont blancs. Le seuil à prendre en compte pour l'extraction des contours sera un attribut de la classe `Outline` nommé `threshold`. Cet attribut devra être initialisé par un constructeur `Outline(double threshold)`.

3.3.1 Tâche 5 : classe `Outline`

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

Changer les instructions de la méthode `initialize` à l'intérieur de la classe `Display` de sorte que vous appliquiez la transformation `Outline` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `threshold` de l'instance de `Outline` devra être égal à `0.055`.

Vous devriez obtenir l'affichage suivant :

3.4 Pixelisation

L'idée de cette transformation est de découper l'image en carré 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 carré pour qu'ils aient ce niveau de gris. La taille d'un coté d'un tel carré sera défini par un attribut `newPixelSize` de la classe `Pixelate`.

3.4.1 Tâche 6 : classe `Pixelate`

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

Changer les instructions de la méthode `initialize` à l'intérieur de la classe `Display` de sorte à ce que vous appliquez la transformation `Pixelate` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). La propriété `newPixelSize` de l'instance de `Pixelate` devra être égal à `10`.

Vous devriez obtenir l'affichage suivant :

3.5 Composition de transformations

L'idée de cette transformation est de composer plusieurs transformations et de les appliquer successivement à l'image. Cette transformation devra donc appliquer une séquence de transformations qui sera un tableau `Transform[] transforms` passé au constructeur.

3.5.1 Tâche 7 : classe `CompositeTransform`

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

Changer les instructions de la méthode `initialize` à l'intérieur de la classe `Display` de sorte que vous appliquiez la transformation `CompositeTransform` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'instance de `CompositeTransform` devra effectuer les trois transformations suivantes (dans cet ordre) :

— `DecreaseGrayLevels` avec `8` pour la valeur de `nbGrayLevels`
— `Outline` avec `0.05` pour la valeur de `threshold`
— `Invert`

Vous devriez obtenir l'affichage suivant :

4 Tâches supplémentaires optionnelles

4.1 Miroir

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

4.2 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

4.3 Menu

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