Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Programmation avancée : TP 6
Code UE : SBICU05

Année 2023-24

Manipulation d'images

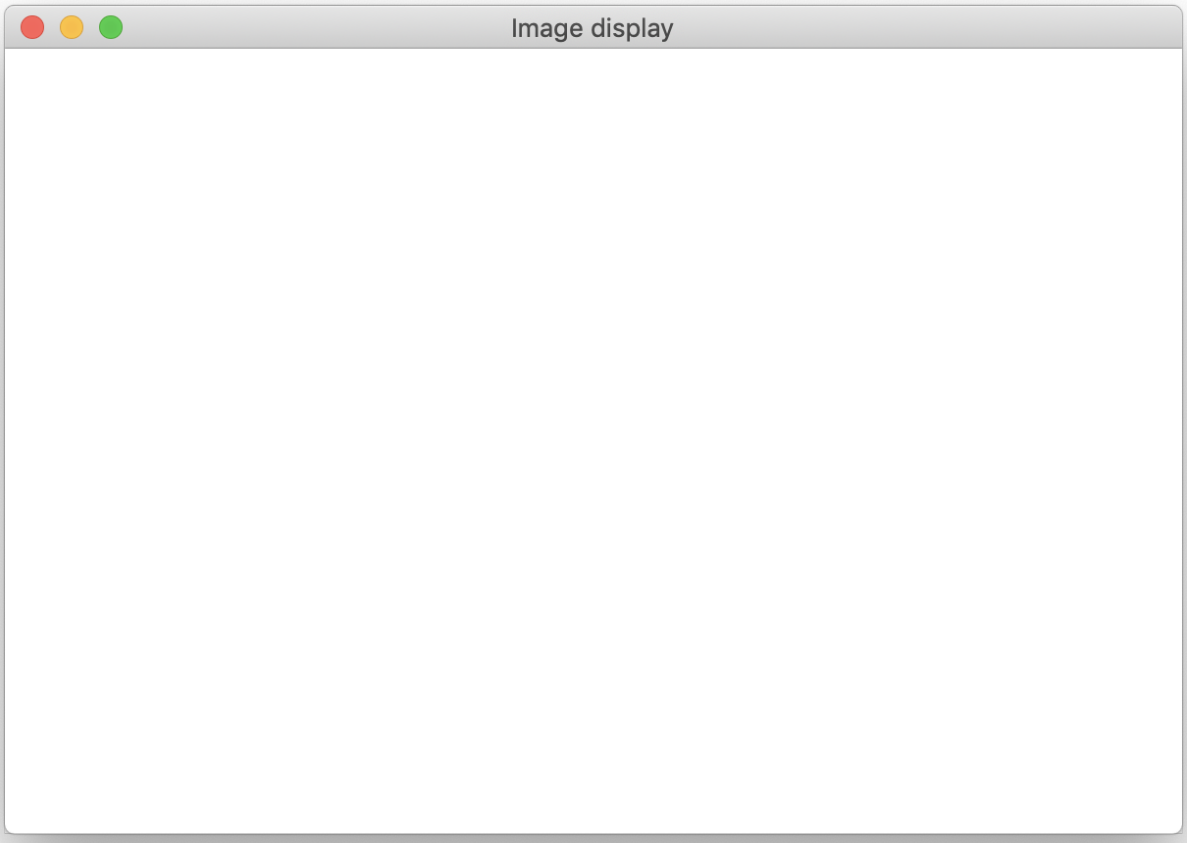
Manipulation d'images

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.

Consignes

Comme pour le TP 2, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du TP 2. Le lien vers le projet à forker est le suivant : lien

Vous pouvez ensuite compiler et exécuter le programme en cliquant deux fois sur `image` -> `application` -> `run` (en n'oubliant pas d'ajouter votre projet en tant que projet *Gradle*). Vous devriez obtenir l'affichage suivant.



Pour exécuter les tests, il faut passer par l'onglet *gradle* à droite et cliquer deux fois sur 'image -> Tasks -> verification -> test. Pour le moment, les tests sont désactivés, car le code des classes sont incomplètes.

Définitions des couleurs et des images

Définition des couleurs

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

```
1 public interface GrayColor extends Comparable<GrayColor> {
2     double getLuminosity();
3     Color getColor();
4 }
```

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 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. Cette méthode est déjà implémentée.
— `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`.

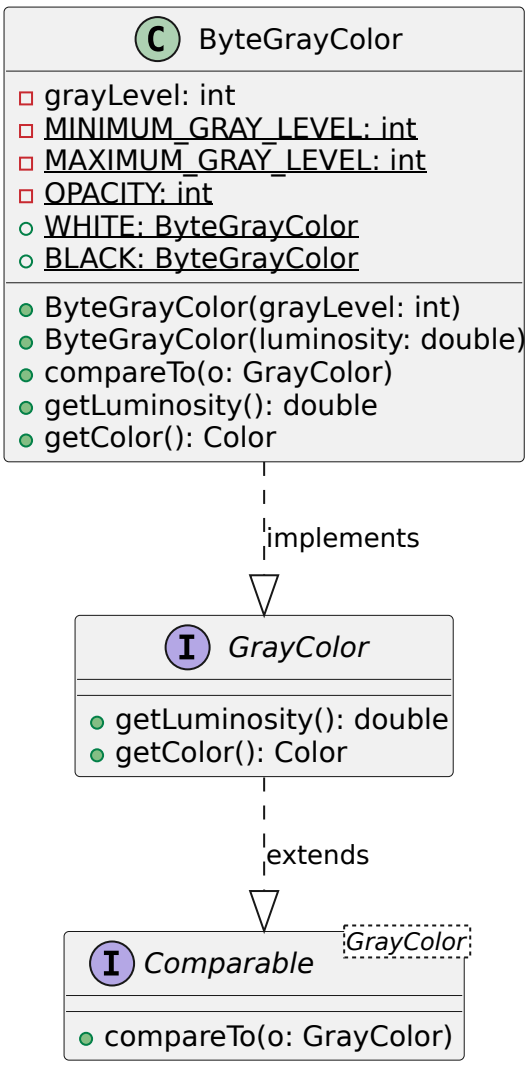
Tâche 1 : classe `ByteGrayColor`

Compléter la classe ByteGrayColor implémentant l'interface GrayColor. La méthode `Color getColor()` est déjà implémentée. En plus de compléter les deux méthodes `double getLuminosity()` et `compareTo(GrayColor o)`, il vous faudra compléter les deux constructeurs suivants (le constructeur `ByteGrayColor()` qui construit une couleur avec un niveau de gris égal à MINIMUM_GRAY_VALUE étant déjà implémenté) :

— `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.

Ajouter aussi deux constantes publiques pour les couleurs noir et blanc que vous nommerez respectivement BLACK et WHITE.

Nous avons donc le diagramme de classe ci-dessous :



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

Définition des images

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

```
1  public interface GrayImage extends Image {
2      void setPixel(GrayColor gray, int x, int y);
3      GrayColor getPixelGrayColor(int x, int y);
4  }
```

L'interface `Image` étant définie par :

```
1  public interface Image {
2      Color getPixelColor(int x, int y);
3      int getWidth();
4      int getHeight();
5  }
```

Une classe implémentant cette interface doit donc implémenter les cinq méthodes suivantes :

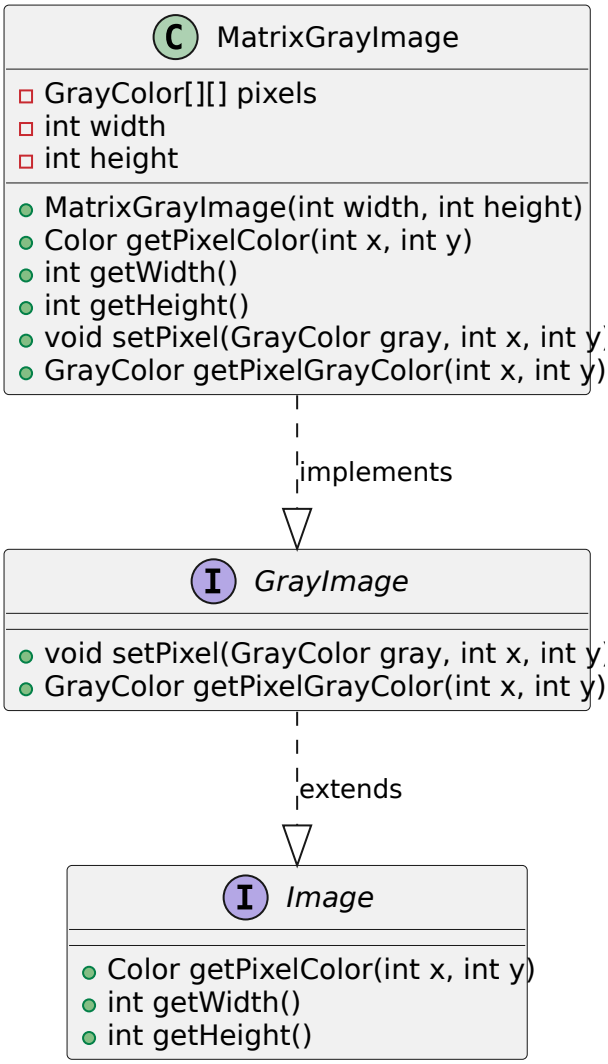
— `GrayColor getPixelGrayColor(int x, int y)` : renvoie la `GrayColor` du pixel (x, y).
— `Color getPixelColor(int x, int y)` : renvoie la couleur du pixel (x, y) pour l'affichage.
— `void setPixel(GrayColor gray, int x, int y)` : change la couleur du pixel (x, y).
— `int getWidth()` : renvoie la largeur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

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.

Nous avons donc le diagramme de classe ci-dessous :



Vous devriez obtenir l'affichage suivant :

Transformations d'images

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

```
1 public interface Transform {
2     void applyTo(GrayImage image);
3 }
```

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

Vous allez donc définir des classes implémentant cette interface.

Inversion des niveaux de gris

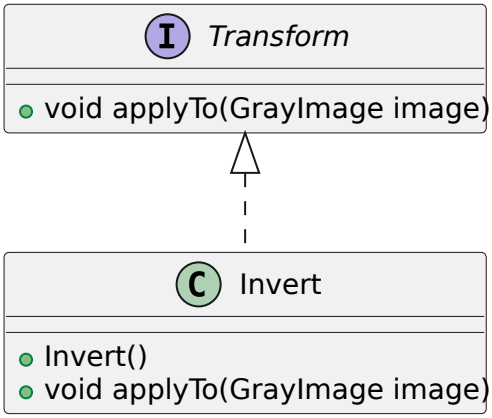
La première transformation consiste à modifier chaque pixel de l'image de sorte que le nouveau niveau de gris de chaque pixel soit égal au niveau de gris maximum auquel on soustrait l'ancien niveau de gris.

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()`.

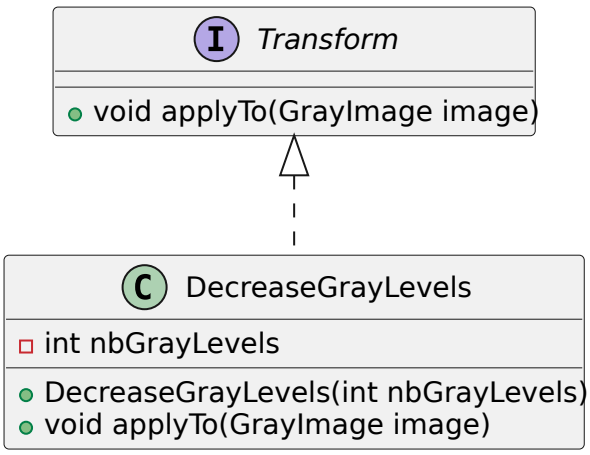


Changer les instructions de la méthode `initialize` à l'intérieur de la classe `Display` de sorte que vous appliquiez la transformation `Invert` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`).

Vous devriez obtenir l'affichage suivant :

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 un attribut `nbGrayLevels` de la classe `DecreaseGrayLevels` implémentant `Transform`, qu'on supposera être un entier.



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 parties : $\left[0, \frac{1}{4}\right[$, $\left[\frac{1}{4}, \frac{2}{4}\right[$, $\left[\frac{2}{4}, \frac{3}{4}\right[$ et $\left[\frac{3}{4}, 1\right[$. Cela nous donne au total 5 intervalles pour l'intervalle $[0, 1]$ puisqu'on y ajoute l'intervalle singleton $[1, 1]$. Une couleur ayant une luminosité dans un de ces 4 intervalles est transformée en la couleur correspondant à la luminosité du début de l'intervalle alors qu'une couleur de luminosité 1 reste de luminosité 1. 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 $0,5$. Vous devez d'ailleurs tester cette propriété en rajoutant une classe de test.

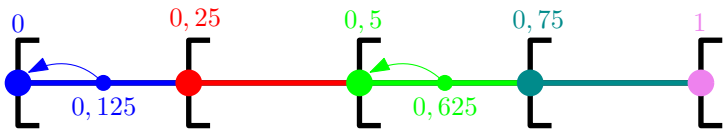


FIGURE 1 – Passage à 5 niveau de gris

tp-image-decreasegreylevels.pdf

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

Tâche 4 : classe `DecreaseGrayLevels`

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

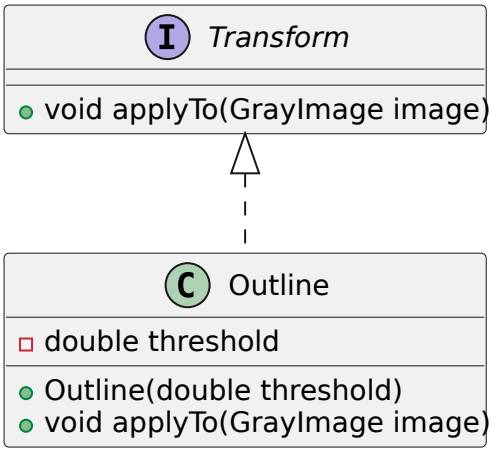
Changer la méthode `initialize` à l'intérieur de la classe `Display` de sorte que vous appliquez la transformation `DecreaseGrayLevels` à l'attribut `image` entre le chargement de l'image (appel à `createImageFromPGMFile`) et son rendu (appel à `render`). L'attribut `nbGrayLevels` de l'instance de `DecreaseGrayLevels`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

Vous devriez obtenir l'affichage suivant :

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)`.

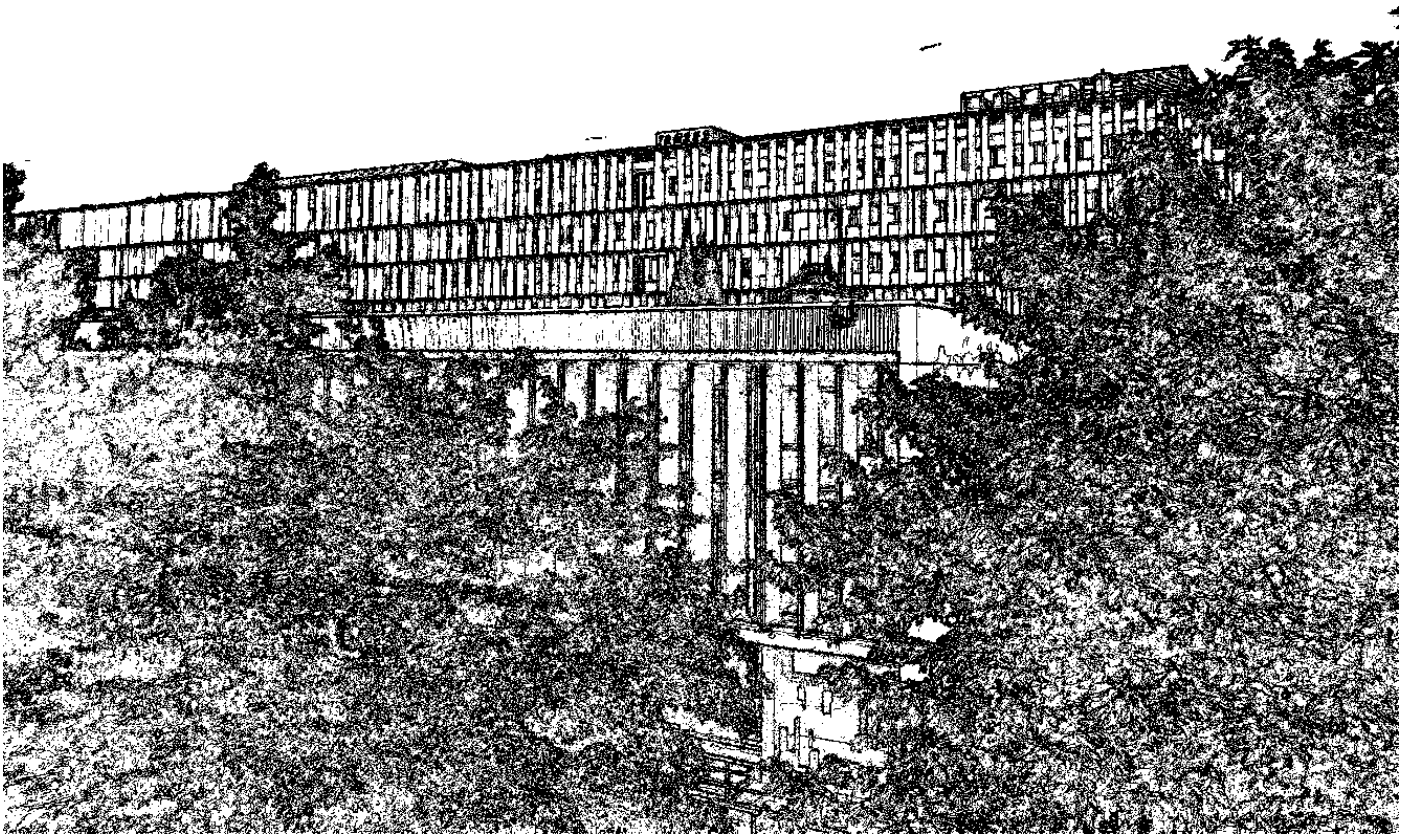


Tâche 5 : classe `Outline`

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 `0.025`.

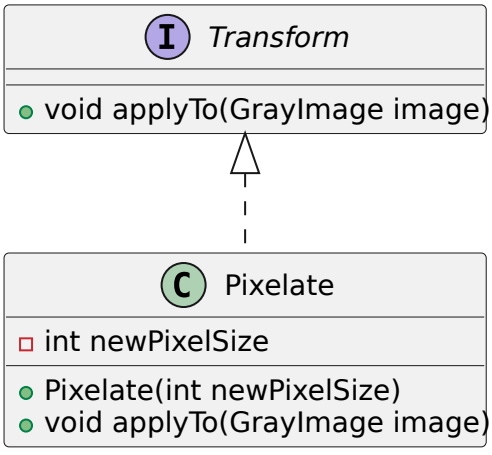
Vous devriez obtenir l'affichage suivant :



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 côté d'un tel carré sera défini par un attribut `newPixelSize` de la classe `Pixelate`.



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 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 :

Composition de transformations

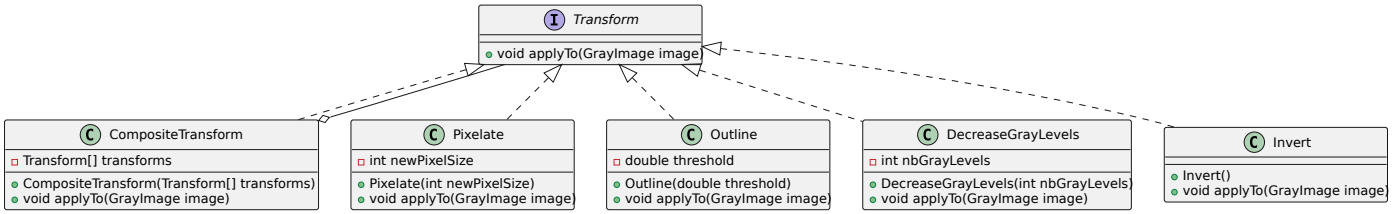
L'idée de cette transformation est de composer plusieurs transformations et de les appliquer successivement à l'image. Cette transformation devra donc appliquer une séquence de transformations qui sera un tableau `Transform[] transforms` passé au constructeur.

Tâche 7 : classe `CompositeTransform`

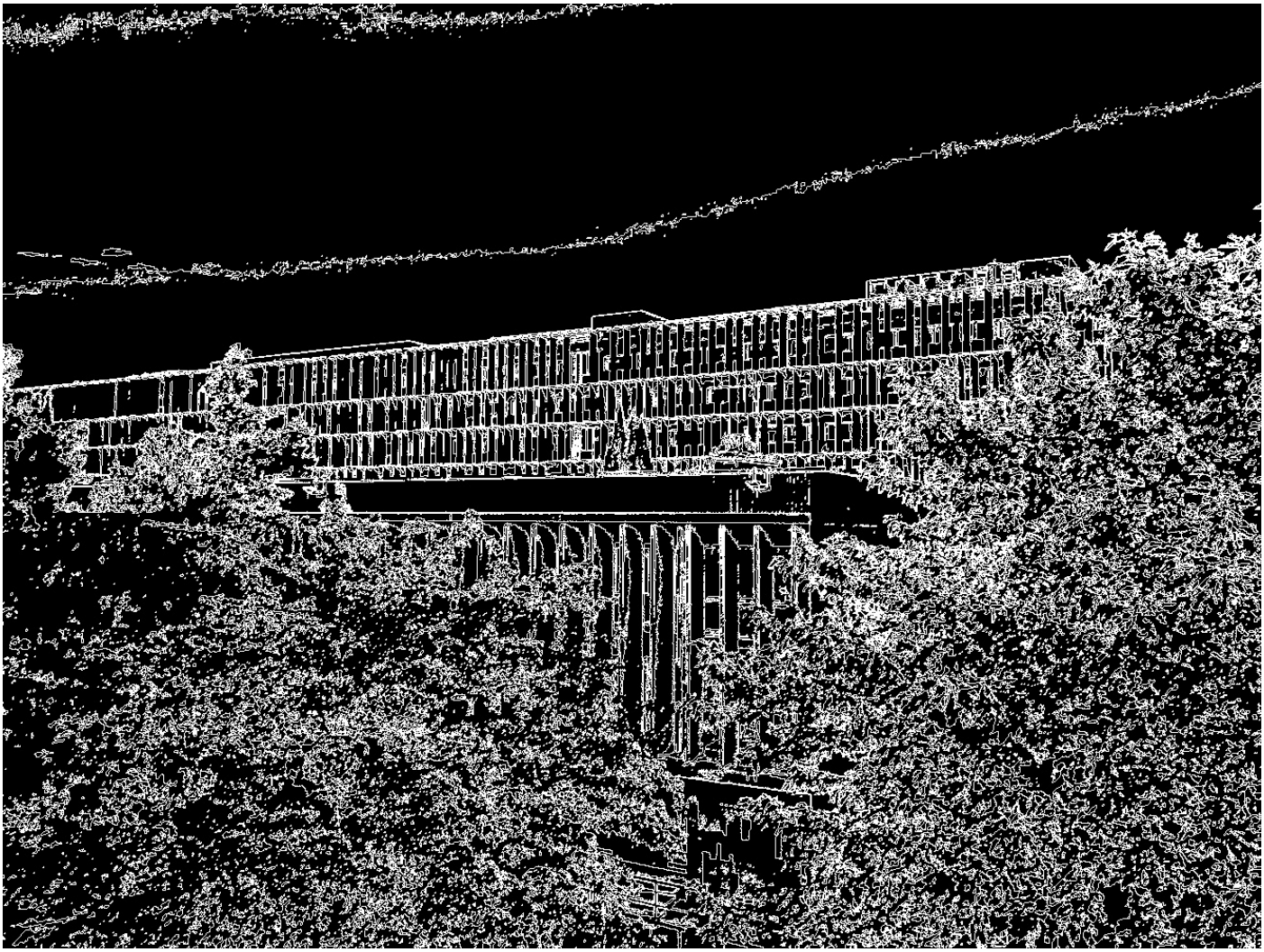
Créer la classe `CompositeTransform` implémentant l'interface `Transform`.

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 :



Tâches supplémentaires

Miroir

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

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

Menu

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