TM7 : Écriture de Fichiers et Dessin d'Images

Le but de ce TP est de se familiariser avec deux notions : la première est la création de fichier par programme, et la seconde qui va de paire est la notion de format de fichiers. Pour se faire la main avec ces concepts, nous allons prendre le cas d'un format de fichier permettant de décrire des images en niveau de gris : le format PGM. À la fin de ce TP, il devrait vous être relativement facile de regarder d'autre format un peu plus complexe permettant d'avoir des images en couleurs comme le format PPM.

1 Familiarisation avec le format PGM

Voici quelques étapes à suivre afin d'apprendre et vérifier votre compréhension du format PGM.

1. Lire la section suivante : `https://fr.wikipedia.org/wiki/Portable_pixmap#Fichier_ASCII_2`.
2. Ouvrir bloc-note, y copier l'exemple de la section et sauvegarder le tout sous le nom `test.pgm`.
3. Ouvrir le fichier `test.pgm` ainsi créé avec un lecteur d'image. Double-cliquer sur le fichier suffira.
4. Remarquer que le fichier n'utilise en fait que 4 niveau de gris : les valeurs 3, 7, 11 et 15. Modifier le fichier avec bloc-note pour qu'il n'utilise réellement que 4 niveaux de gris. Commencer par changer la valeur 15 se trouvant à la ligne 4. Cette valeur est la valeur maximale du fichier, celle de la couleur blanc. Vérifier qu'après votre modification, le fichier s'affiche correctement.
5. Maintenant votre fichier n'utilise que les valeurs 0, 1, 2, 3 et 4. Que se passe-t-il se l'on met la valeur 64 sur la ligne 4 malgré tout ?
6. Sur la troisième ligne se trouve la largeur et la hauteur de l'image. Pour le moment il s'agit de `24 7`. Nous avons donc $24 \times 7 = 168$ pixels sur notre image. Essayer de prédire se qui va se passer si l'on indique `12 14` sur la ligne 3 puis vérifier se qu'il se passe réellement.
7. Quel est le nombre maximal de caractère par ligne dans le format PGM ? Que se passe-t-il si on ajoute des passages à la ligne après chaque valeur du fichier ?

2 Création d'images PGM avec bloc-note

Maintenant, créons quelques fichiers à la main. Si nous ne savons pas le faire à la main, comment pourrions-nous le programmer ?! Comment "expliquer à l'ordinateur" ce que nous ne savons pas faire ?!

1. Faire entièrement à la main l'image FEEP précédente mais avec le texte écrit de haut en bas.
2. Faire une image 20×20 sur fond gris avec, en plein centre, un carré blanc de 10 pixels de coté.
3. Dans l'image précédene, si l'on numérote les colonnes de 0 à 19 (cela fait bien 20 colonnes), quelles sont les numéros des colonnes gauche et droite du carré ? Qu'en est-il des lignes supérieure et inférieure ?
4. Faire un programme qui demande à l'utilisateur une taille t pour l'image et une taille c pour le carré et qui affiche dans la console ce qui doit être écrit dans le fichier (tout du P2 initiale à la fin du fichier) pour avoir une image de taille $t \times t$ avec un carré de coté c en plein centre.
   Conseil : Commencer par un `clear()` pour vous simplifier la vie. Décider vous même comment gérer le sens de "en plein centre" pour les différentes parités possible de t et de c. Tester votre programme pour créer une image de taille 300×300 avec un carré de taille 100.

3 Création directe d'images PGM

Nous souhaitons maintenant créer directement des fichiers PGM sans avoir à afficher dans la console puis à copier-coller. Il nous faut pouvoir indiquer le nom du fichier à créer, avoir des équivalents de `print` et `println` qui "affiche dans le fichier", et enfin pouvoir dire que le fichier est terminé et peut être sauvegarder. Voici un programme simple qui montre le nécessaire. D'abord on crée un "parcours" vers un fichier. Ensuite on crée un "writer" Pour écrire une nouvelle ligne on utilise `"\n"`. Le "main" doit déclarer une exception (on vous expliquera cela plus tard.)

```
import java.nio.file.*;
import java.io.*
public class App {
        public static void main(String[] args) throws IOException {
                Path file = Paths.get("monfichier.txt");
                BufferedWriter writer = Files.newBufferedWriter(file);
                writer.write("Voici quelques");
                writer.write(" mots.\n");
                writer.write("Voici quelques");
                writer.write(" lignes.\n");
                writer.close();
        }
}
```

1. Tester le programme ci-dessus en Java
2. Modifier votre programme de dessin de carré afin qu'il crée un fichier au lieu de passer par la console.
3. Étendre votre programme pour que l'image puisse être rectangulaire avec une taille et une couleur de fond choisi par l'utilisateur et que le carré soit maintenant un rectangle dont la taille, la position et la couleur sont indiquées par l'utilisateur. Choisir la façon dont ces informations sont indiqués par l'utilisateur. Il y a plusieurs façons de faire.

4 Structurer un programme plus complexe

Jusqu'ici, il n'était pas strictement nécessaire d'organiser son code avec des fonctions ni d'organiser ces données avec des structures. Maintenant nous souhaitons pouvoir demander à l'utilisateur autant de rectangles qu'il le souhaite et créer le fichier correspondant. Pour cela, nous allons nous organiser un peu. Nous allons utiliser une structure `PGMImage` contenant toutes les informations sur l'image (tailles, valeur du blanc et valeurs des pixels). Nous allons également utiliser une structure `Rectangle` pour représenter un rectangle(taille, position et couleur), ainsi qu'un tableau de rectangle pour mémoriser toutes les demandes de l'utilisateur. Nous ferons donc des petites fonctions qui, combiner ensemble, feront le programme désiré.

1. Proposer une définition pour la structure `PGMImage`.
   Conseil : Utiliser un tableau à deux dimensions pour mémoriser les valeurs des pixels.
   Choisir clairement lequel des deux indices indique la ligne, et lequel indique la colonne.
2. Faire une fonction `PGMImage createPGMImage(int width, int height, int white, int back)` qui crée une instance de `PGMImage` avec les informations indiquées, `width` et `height` étant la largeur et la hauteur, `white` étant la valeur maximale et `back` étant la couleur initiale des pixels de l'image.
3. Faire une fonction `void savePGMImage(PGMImage img, String filename)` qui prend une `PGMImage` et crée le fichier correspond avec le nom `filename`.
4. Faire une fonction `Rectangle createRectangle(...)` qui crée un rectangle sur la base des informations que vous avez choisi. Rappel : il ne doit y avoir aucun `readQuelqueChose` dans cette fonction.
5. Faire une fonction `void drawRectangle(PGMImage img, Rectangle rect)` qui dessine le rectangle correspondant aux données contenues dans `rect` en modifiant les valeurs des pixels de l'image `img`.
6. Faire une fonction `void drawRectangles(PGMImage img, Rectangle[] rects)` qui utilise la fonction précédente pour dessiner tous les rectangles du tableau `rects` sur l'image `img`.

Nous avons fait toutes ces fonctions sans interactions avec l'utilisateur. C'est toujours une bonne idée de faire comme cela. Cela permet par exemple d'automatiser les tests des fonctions sans avoir à saisir des valeurs de test en permanence (avec les possibles erreurs de saisi à répétition qui vont avec) et ce n'est qu'un des nombreux avantages ! Nous allons maintenant ajouter les interactions avec l'utilisateur.

7. Faire une fonction `PGMImage readPGMImage()`
8. Faire une fonction `Rectangle readRectangle()`
9. Faire une fonction `Rectangle[] readRectangles()` qui utilise la fonction précédente pour demander à l'utilisateur de saisir autant de rectangle qu'il le souhaite.
10. Faire le programme final qui combine toute ses fonctions pour créer une image de rectangles selon les désirs de l'utilisateur.
11. Refaire un passage sur tous le programme et choisir les noms des variables de façon à ce que le code se lise comme un roman. Choisir également l'ordre des définitions des structures et des fonctions de façon à faciliter la lecture du code.

5 Lecture et modification d'images PGM

Le but de cette section est de pouvoir programmer des manipulations d'image comme pivoter une image de 90 degrée, inverser les couleurs, ou combiner deux images en indiquant une couleur de "transparence" sur la deuxième image. Pour cela, il faut déjà réussir à lire le contenu d'un fichier PGM. Pour cela, de façon similaire à l'écriture d'un fichier, il faut pouvoir indiquer le fichier à lire, avoir des équivalents de readInt, readString et compagnie pour lire dans le fichier et pouvoir indiquer lorsque le lecture est terminer et que le fichier peut être fermé. Voici un programme simple qui lit un fichier.

```
import java.io.*
import java.util.*;
import java.nio.file.*;
public class App {
        public static void main(String[] args) throws IOException {
                Path file = Paths.get("monfichier.txt");
                Scanner reader = new Scanner(file);
                int numberOfValue;
                numberOfValue = reader.nextInt();
                int sum = 0;
                for (int i = 0; i < numberOfValue; i = i + 1) {
                        int value = reader.nextInt();
                        sum = sum + value;
                }
                System.out.println(sum);
                reader.close();
        }
}
```

1. Lire le programme précédent pour déterminer quelle genre de fichier il lit.
2. Écrire un fichier de test avec tous les nombres sur la même ligne et un autre les nombres sur plusieurs lignes. Tester le programme sur les deux fichiers.
3. Tester le programme précédent : attention, si le fichier n'existe pas ou n'est pas trouvé à cause de son répertoire, Javascool peut bloquer !
4. Tester le programme précedent avec tous les nombres sur la même

Revenons maintenant sur votre programme précédent de dessin.

5. Faire la fonction `PGMImage readSimplePGMFile(String filename)` qui lit un fichier PGM ne contenant aucun commentaire (et donc ne contenant que P2 suivit de plein de nombres). Ce fichier devra au moins pouvoir lire les fichiers créés avec votre fonction `savePGMFile(...)`.
6. Faire la fonction `PGMImage rotate90Right(PGMImage img)` qui crée une nouvelle instance de `PGMImage` correspondant à l'image `img` retourner de 90 degrées vers la droite.
7. Faire la fonction `PGMImage flipHorizontal(PGMImage img)` qui crée une nouvelle instance de `PGMImage` correspondant à l'image `img` avec la gauche et la droite inversées.
8. Faire la fonction `PGMImage flipVertical(PGMImage img)` qui crée une nouvelle instance de `PGMImage` correspondant à l'image `img` avec le haut et le bas inversés.
9. Faire la fonction `PGMImage combine(PGMImage img1, PGMImage img2, int transparent)` qui crée une nouvelle instance de `PGMImage` correspondant à l'image `img1` sur laquelle est dessiner l'image `img2` en considérant les pixels de couleur `transparent` de `img2` comme transparent.
10. Faire un programme qui propose un menu à l'utilisateur afin de lui demande quelles actions ils souhaitent faire et sur quelle(s) fichier(s).