NFA032 : TP 3
Conception de classes et manipulation d'images

2 mars 2020

Les objectifs de ce TP dont le sujet est la manipulation d'images au format PPM sont les suivants :

- réfléchir à la conception de classes (qu'est-ce qu'une image, qu'est-ce qu'un pixel ? De quelles méthodes doit on doter les classes représentant ces notions ?)
- utiliser une classe prédéfinie (la classe `PPM` pour la lecture et l'écriture d'un fichier image au format PPM)
- commencer à manipuler des images et réfléchir à plusieurs représentations des images en vue de se préparer au projet

À la fin du TP, vous devrez remettre une archive zip sur votre espace numérique de formation (ENF, sur le site `www.lecnam.net`, comme vous l'avez fait lors des précédents TPs).

1 Préliminaires : représentation des images et format PPM

Ce TP va vous conduire à effectuer certains traitements sur des images déjà existantes ou que vous créerez vous-mêmes. Ce paragraphe décrit comment une image est représentée d'un point de vue logique, et présente un des nombreux formats existants pour stocker une image sur disque : le format de fichier `PPM` utilisé dans ce TP.

1. **Qu'est-ce qu'une image numérique ?** Une image peut être considérée comme une grille de points appelés **pixels** ayant chacun sa propre couleur. Une couleur peut être décrite comme un mélange de rouge, de vert et de bleu. Le nombre de pixels par ligne est appelé la largeur de l'image et le nombre de lignes sa hauteur. Chaque pixel de l'image est caractérisé par un niveau de rouge (un entier r compris entre 0 et 255), un niveau de vert (un deuxième entier v compris entre 0 et 255) et un niveau de bleu (entier b entre 0 et 255). Chaque triplet (r, v, b) détermine une couleur. Il y a donc $256 \times 256 \times 256$ couleurs pouvant être représentée (plus de 16 millions). Par exemple, la couleur noire correspond au triplet $(0, 0, 0)$, le triplet $(0, 255, 0)$ correspond à du bleu, le triplet $(255, 255, 0)$ à du jaune, etc..
2. **Le format de fichier PPM.** De très nombreux formats de fichiers d'images existent (jpeg, gif, png, ...). Nous utiliserons pour ce TP un format peu efficace mais simple à comprendre et à manipuler : le format PPM. Ce format est décrit au paragraphe **??**. Vous n'aurez toutefois pas besoin de connaître la syntaxe utilisée par les fichiers PPM pour effectuer ce TP, dans la mesure où vous utiliserez une classe fournie pour l'occasion pour lire ou écrire un fichier PPM : la classe PPM. Sous Linux, vous visualiserez une image au format PPM à l'aide de la commande `display` et sous windows avec le logiciel gratuit `Gimp`. Vous trouverez sur le site du cours un exemple de fichier PPM, le fichier `perruche.ppm`.
3. **La classe PPM fournie pour ce TP**. Son code est fourni au paragraphe **??**. Cette classe comporte uniquement deux méthodes statiques :
   - la fonction `lireFichier()` dont la signature est :

```
public static int[][][] lireFichier(String cheminFichier)
```

Cette fonction prend pour unique paramètre le chemin d'accès à un fichier image et se charge de la lecture du fichier. Elle retourne un tableau d'entiers à trois dimensions (donc de type `int[][][]`) représentant l'image qui a été lue. Plus précisément, si `t` désigne le tableau retourné par cette fonction :

- `t[i][j][0]` désigne le niveau de rouge du pixel (i,j) de l'image
- `t[i][j][1]` désigne le niveau de vert du pixel (i,j) de l'image
- `t[i][j][2]` désigne le niveau de bleu du pixel (i,j) de l'image
- `t.length` désigne la hauteur de l'image (son nombre de lignes)
- `t[0].length` désigne la largeur de l'image (son nombre de colonnes)

— la méthode statique `ecrireFichier()` utilise deux paramètres : le chemin d'un fichier à écrire et un tableau d'entiers à 3 dimensions représentant une image. La méthode se charge de créer un fichier au format PPM correspondant au tableau passé en paramètre. L'en-tête de cette méthide est :

```
public static void ecrireFichier(String cheminFichier, int[][][] t)
```

2 Le traitement à réaliser

On désire écrire un programme qui lit un fichier image au format PPM (le fichier `perruche.ppm`), réalise plusieurs traitements sur l'image, et écrit un ou plusieurs fichiers correspondant à l'image après traitement. Les traitements à réaliser sont les suivants :

— Transformer l'image en noir et blanc : chaque pixel est remplacé par le triplet (m, m, m) où m désigne la moyenne de ses niveaux de rouge, vert et bleu :

Image en noir et blanc

— L'image originale comporte une perruche verte et une perruche bleue. Le premier traitement consiste à transformer la perruche verte en canari (en rendant jaunes tous les pixels de l'image comportant plus de 40% de vert) et à rendre la perruche bleue violette (en ajoutant du rouge à chaque pixel de l'image comportant plus de 35% de bleu) :

Image originale

Image transformée

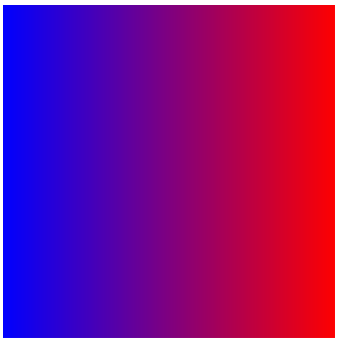
— Doubler la taille de l'image : transformer chaque pixel en 4 pixels de la même couleur.

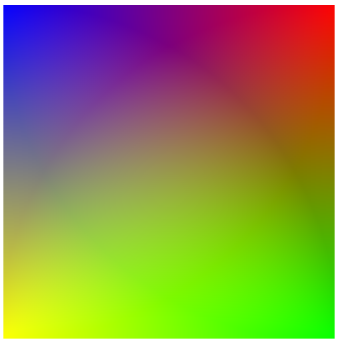
3 Déroulement du TP

1. Créez un projet java dans Eclipse nommé `Tp3`.
2. On désire manipuler des images. Plusieurs représentations sont possibles. On vous demande dans ce TP de concevoir deux classes :
   — une classe `Pixel` permettant de représenter et manipuler un pixel.
   — une classe `Image` permettant de représenter et manipuler une image. Une image sera vue comme un tableau à **deux** dimensions de pixels.
3. Réfléchissez à la conception de ces classes : de quelles variables et de quelles méthodes doit-on les doter en vue de réaliser les traitements à effectuer ?
4. Écrivez ces deux classes. Il conviendra que la classe `Image` propose notamment un constructeur prenant en paramètre le chemin d'un fichier sur le disque, par exemple :
   "C : \\Users\\Eric\\Desktop\\perruche.ppm" ou "/home/Eric/perruche.ppm"
   Ce constructeur utilisera la méthode statique `PPM.lireFichier()` pour lire le fichier dont le nom est passé en paramètre.
   Dotez vos deux classes d'une méthode `toString()`.
5. Dans une autre classe (`Main`), écrivez une méthode `main()`. Créez dans le `main()` un objet de votre classe `Image` correspondant au fichier `perruche.ppm` que vous trouverez sur le site du cours au moyen du constructeur de la classe Image.
6. Effectuez l'un des traitements envisagés au paragraphe précédent. Il est conseillé de choisir la mise en noir et blanc.
7. Utilisez la méthode statique `PPM.ecrireFichier()` pour écrire un fichier que vous pourrez nommer `sortie.ppm` par exemple, représentant l'image transformée. Affichez cette image à l'écran (avec Gimp ou `display`)
8. Créez une archive zip et déposez la sur votre ENF en utilisant le formulaire de remise du Tp3.

4 Pour aller plus loin...

— Nous vous proposons de réaliser les autres traitements mentionnés dans le paragraphe **??**.
— Vous pouvez également ajouter des méthodes aux classes que vous avez écrites, notamment pour permettre de créer des images représentant des dégradés de couleurs :





5 Annexe 1 : le format PPM

Un fichier PPM est composé de la façon suivante. Sur la première ligne, il y a écrit **P3**. Sur la deuxième ligne il y a écrit le caractère # suivi d'un commentaire, par exemple le nom du fichier. Sur la troisième ligne, il y a la largeur et la hauteur de l'image, séparées par un espace, sur la quatrième ligne, il y a un nombre entier qui est le plus grand nombre utilisable dans la représentation des couleurs. C'est généralement 255 (ou bien une puissance de 2 moins 1). Supposons que ce soit le nombre 255. Sur les lignes suivantes, il y a la couleur de chaque point donnés par 3 nombres compris entre 0 et 255, séparés par un espace (ou plusieurs espaces) représentant respectivement la quantité de rouge, de vert et de bleu présents dans le mélange. Les fins de ligne du fichier sont considérées comme de simples espaces. Les lignes doivent en principe comporter moins de 70 caractères.

Pour dire qu'un point est rouge, on donnera les valeurs 255 0 0. Cela donne la couleur obtenue en mélangeant le maximum de rouge (255) avec pas du tout de vert ni de bleu. Le vert s'obtient avec les nombres 0 255 0. Le bleu avec 0 0 255. Le jaune avec 255 255 0. Le noir avec 0 0 0. Le blanc avec 255 255 255. En mettant moins de 255, par exemple, 100 100 0, on obtient un jaune plus foncé.

Les points sont donnés dans l'ordre suivant : de gauche à droite et de haut en bas. Le premier point donné est celui qui est en haut à gauche, puis le deuxième point de la première ligne et ainsi de suite jusqu'au point le plus à droite de la première ligne. Ensuite viennent les points de la seconde ligne, etc.

Exemple de fichier image

```
P3
#truc.ppm
4 4
255
0     0 0    0   0  0    0   0  0   255  0  0
0     0 0    0 255 70    0   0  0     0  0  0
0     0 0    0   0  0    0 255 70     0  0  0
255 255 0    0   0  0    0   0  0     0  0  0
```

On a une image de taille 4 points par 4 points (4 lignes, 4 colonnes). Il y a donc 16 points en tout. Chaque point est décrit par 3 entiers, il y a donc 16x3=48 nombres entre 0 et 255. Les trois

premiers points de la première ligne sont noirs. Le dernier est rouge. Le quatrième point de la première colonne est jaune.

Vous pouvez taper ce fichier avec un éditeur de texte (nedit, edit). Il faut lui donner le nom truc.ppm. Vous pouvez ensuite voir l'image en utilisant un visionneur d'images (gimp sous windows). Sur les machines du Cnam et la plupart des distributions Linux, la commande `xv` permet de visualiser les images au format PPM.

L'image donnée en exemple est toute petite. Il est très long de taper à la main le contenu d'une grande image, d'où l'intérêt d'un programme qui manipule des images déjà existantes (vous en trouverez une sur le site web du cours).

6 Annexe 2 : la classe PPM

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.FileNotFoundException;
import java.io.IOException;

class PPM {

  /**
   * Lit un fichier texte au format PPM et renvoie l'image correspondante sous la forme
   * d'un tableau d'entiers à trois dimensions. Si t désigne le tableau retourné,
   * t[i][j][0] représente le niveau de rouge du pixel (i,j) de l'image
   * t[i][j][1] représente le niveau de vert du pixel (i,j) de l'image
   * t[i][j][2] représente le niveau de bleu du pixel(i,j) de l'image
   * Enfin, le nombre de lignes de l'image est t.length et le nombre de colonnes t[0].length
   * @param cheminFichier : chemin du fichier à écrire
   *                        ex windows : "C:\\Users\\Eric\\Desktop\\sortie.ppm"
   *                        ex linux : "/home/Eric/sortie.ppm"
   * @param t :
   */
    public static int[][][] lireFichier(String cheminFichier) {
        BufferedReader lecteur=null;
        String ligne;
        int nbCol = 0, nbLig = 0, maxP = 255;
        int cpt;
        int []tabValeurs = new int[0];

        try {
            lecteur = new BufferedReader(new FileReader(cheminFichier));
            ligne = lecteur.readLine(); // lecture du symbole P3
            nbCol = Integer.parseInt(lecteur.readLine());
            nbLig = Integer.parseInt(lecteur.readLine());
            maxP  = Integer.parseInt(lecteur.readLine());
            tabValeurs = new int[3+nbLig*nbCol*3];
            tabValeurs[0] = nbCol;
            tabValeurs[1] = nbLig;
            tabValeurs[2] = maxP;
            cpt = 3;

            while (((ligne = lecteur.readLine())!=null) && (cpt < 3+nbLig*nbCol*3)){
             tabValeurs[cpt] = Integer.parseInt(ligne);
```

```
            cpt++;
          }
          lecteur.close();
      }
      catch(FileNotFoundException exc) {
          System.out.println("Erreur d'ouverture");
      }
      catch(IOException e) {
       System.out.println("Erreur lecture de ligne");
      }

  int[][][] t = new int[nbLig][nbCol][3];
  int k = 3, nbPixParLigne = 0, numLigne = 0, numCol = 0;
  while (k < tabValeurs.length) {
    t[numLigne][numCol][0] = tabValeurs[k];
    t[numLigne][numCol][1] = tabValeurs[k+1];
    t[numLigne][numCol][2] = tabValeurs[k+2];
    k +=3;
    nbPixParLigne++;
    if (nbPixParLigne == nbCol) { numLigne++; numCol = 0; nbPixParLigne = 0; }
    else numCol++;
  }
  return t;
  }


/**
 * Ecrit un fichier texte au format PPM à partir d'un tableau contenant toutes les
 * valeurs entières à écrire dans le fichier après le code P3. Les 3 premières valeurs
 * du tableau sont le nombre de colonnes, le nombre de lignes et la valeur maximale de
 * chaque couleur (ex : 255) puis la succession des niveaux r v b de chaque pixel
 * @param cheminFichier : chemin du fichier à écrire
 *                        ex windows : "C:\\Users\\Eric\\Desktop\\sortie.ppm"
 *                        ex linux : "/home/Eric/sortie.ppm"
 * @param t :
 */
  public static void ecrireFichier(String cheminFichier, int[][][] t) {
  int nbLig = t.length;
  int nbCol = t[0].length;
  BufferedWriter ecrivain=null;

  try {
    ecrivain = new BufferedWriter(new FileWriter(cheminFichier));
          ecrivain.write("P3\n"+nbCol+"\n"+nbLig+"\n"+"255\n");
          for(int i=0; i<nbLig; i++) {
              for (int j = 0; j < nbCol; j++) {
               ecrivain.write(""+t[i][j][0]+"\n");
               ecrivain.write(""+t[i][j][1]+"\n");
               ecrivain.write(""+t[i][j][2]+"\n");
             }
          }
          ecrivain.close();
      }
      catch(FileNotFoundException exc) {
          System.out.println("Erreur d'ouverture");
      }
```

```
    catch(IOException e) {
           System.out.println("Erreur écriture de ligne");
        }
    catch(IndexOutOfBoundsException e) {
      System.out.println("Erreur : le tableau n'est pas au bon format.");
    }
  }
}
```