1 Les images et les niveaux de gris

Dans ce projet, nous allons chercher à écrire plusieurs classes permettant de représenter des images en niveau de gris que nous visualiserons soit en mode texte, soit en mode graphique.

Une image noir et blanc peut être représentée comme un rectangle de dimensions `largeur` × `hauteur`, contenant des points caractérisés par leur intensité lumineuse ou *niveau de gris*.

Nous avons choisi de représenter de façon abstraite ces niveaux de gris, pour pouvoir, d'une part, les représenter soit en mode texte (par des caractères), soit en mode graphique (par des couleurs) ; d'autre part, leur appliquer un certain nombre d'opérations (ajout, soustraction, inversion, etc.).

On décide ensuite de construire une `ImageGrise` à partir de ces niveaux de gris, en respectant l'interface suivante :

ImageGrise.java

```
package image ;
/** Cette interface représente de façon abstraite une image en niveaux de gris.
 * <BR>Toute image est caracterisée par ses dimensions (largeur et hauteur) et
 * par les niveaux de gris associés à chaque coordonnée (x, y).<P>
 * Par défaut, une image nouvellement créée doit être entièrement <B>blanche</B> */
public interface ImageGrise {
    /** Retourne la largeur de l'image */
    int largeur() ;
    /** Retourne la hauteur de l'image */
    int hauteur() ;
    /** Retourne le niveau de gris du point de coordonnées (x,y) */
    NiveauGris pointEn(int x, int y) ;
    /** Fixe le niveau de gris du point de coordonnées (x,y) à la valeur spécifiée */
    void definirPoint(int x, int y, NiveauGris gris) ;
    /** Met en noir le point de coordonnées (x,y) */
    void allumer(int x, int y) ;
    /** Met en blanc le point de coordonnées (x,y) */
    void eteindre(int x, int y) ;
    /** Donne une valeur aléatoire (noir ou blanc) à chaque point de l'image */
    void randomize() ;
    /** Compte le nombre de points de l'image dont le niveau de gris est égal au niveau spécifié */
    int compterPoints(NiveauGris gris) ;
    /** Retourne une image qui est le négatif de l'image courante */
    ImageGrise inverser() ;
    /** Retourne une image dont tous les points (sauf blancs) sont un niveau
     * plus clair que dans l'image courante */
    ImageGrise eclaircir() ;
    /** Retourne une image dont tous les points (sauf noirs) sont un niveau
     * plus foncé que dans l'image courante */
    ImageGrise assombrir() ;
    /** Retourne une <B>copie</B> de l'image courante */
    ImageGrise dupliquer() ;
    /** Retourne une image en additionnant point par point les niveaux de gris de l'image
     * courante et de l'image en paramètre (les deux images doivent être de même taille) */
    ImageGrise ajouter(ImageGrise img) ;
    /** Retourne une image en retranchant point par point les niveaux de gris de l'image
     * courante et de l'image en paramètre (les deux images doivent être de même taille) */
    ImageGrise soustraire(ImageGrise img) ;
    /** Retourne une image en faisant un OU Exclusif (XOR) point par
     * point les niveaux de gris de l'image courante et de l'image en
     * paramètre (les deux images doivent être de même taille) */
    ImageGrise XOR(ImageGrise img) ;
    /** Retourne une image qui représente "l'intersection" de l'image courante et de l'image
     * en paramètre : seuls les points qui ont le même niveau de gris dans les deux images sont
     * conservés (les deux images doivent être de même taille) */
    ImageGrise intersection(ImageGrise img) ;
    /** Retourne le niveau de gris moyen de l'image. Pour le calculer, il faut faire la
     * moyenne des niveaux de chaque point de l'image (ce qui revient à compter combien il y
     * a de points de chaque niveau de gris possible) */
    NiveauGris niveauMoyen() ;
    /** Retourne une image obtenue en augmentant le contraste de l'image courante. Pour
     * augmenter le contraste, il faut rendre les points sombres plus sombres qu'ils ne sont,
     * et les points clairs plus clairs. Un bon moyen de procéder consiste à calculer le
     * niveau de gris moyen de l'image, et assombrir (respectivement eclaircir) les points
     * plus sombres (resp. plus clairs) que ce niveau moyen */
    ImageGrise augmenterContraste() ;
}
```

Pour ne pas partir de rien...

Récupérez le fichier `pr4.tgz` et décompressez-le. Vous y trouverez les éléments suivants :

- dans `sources` :
  - un package `image` qui contient les sources de l'interface `ImageGrise` et de la classe `NiveauGris` ;
  - un package `dictionnaire.correction` qui donne le corrigé des classes `TabDict` et `CoupleObj` du TP 4 ;
  - une classe `Catalogue` qui calcule quelques images simples.
- dans `classes` : le *bytecode* d'un `Afficheur` permettant de visualiser les images dans une fenêtre graphique, dans un fichier `afficheur.jar` ;
- dans `doc` : la documentation des fichiers fournis.

2 Réalisation au moyen d'un tableau de niveaux de gris

Q 1. Écrivez une classe `image.ImageTab` implémentant l'interface `image.ImageGrise` au moyen (entre autres) d'un attribut `NiveauGris [][]`. Vérifiez *soigneusement* le fonctionnement de *toutes* les méthodes (par exemple, mais pas exclusivement, en vous servant de la classe `Catalogue`).

3 Réalisation au moyen d'un dictionnaire

On remarque qu'il arrive fréquemment, pour des images simples, qu'il y ait un grand nombre de points blancs. Ces points blancs peuvent d'une certaine façon être considérés comme de la place mémoire perdue, et l'on souhaiterait donc ne mémoriser que la présence des points de gris léger à noir.

Pour cela, on propose d'utiliser un dictionnaire qui mémorisera, pour chaque coordonnée (couple `(x,y)`), le niveau de gris (autre que blanc) qui lui est associé dans l'image. Les points blancs ne correspondent donc à aucune entrée (aucune clef) dans le dictionnaire.

Q 2. Écrivez une classe `image.ImageDict` implémentant l'interface `image.ImageGrise` au moyen d'un attribut de type `Dictionnaire`. Si vous n'avez pas terminé la réalisation de votre propre package `dictionnaire` (TP 4), vous pourrez vous servir du package `dictionnaire.correction` fourni.

Indication : si vous réfléchissez en termes de « briques de base » (méthodes spécifiquement liées à l'implémentation, sur lesquelles on peut s'appuyer pour écrire des méthodes de plus haut niveau), le code de la plupart des méthodes sera en grande partie identique au code de `image.ImageTab` !

Q 3. Vérifiez le fonctionnement de cette nouvelle implémentation en remplaçant `ImageTab` par `ImageDict` dans le fichier `Catalogue.java`.

Q 4. Expliquez (en fonction de la complexité des opérations ou des modes d'accès aux données) les différences de *vitesse d'affichage* qui peuvent être observées entre `ImageTab` et `ImageDict`. Quel(s) mécanisme(s) pourrait-on envisager pour accélérer significativement les traitements les plus lents ? (Donnez la réponse dans les commentaires de la classe `ImageDict`.)

Q 5. *QUESTION SUBSIDIAIRE :*
Pour optimiser l'utilisation de la mémoire, on voudrait pouvoir convertir automatiquement une image implémentée par tableau en une image identique implémentée par dictionnaire, ou vice-versa selon la proportion de points blancs dans l'image.
Ajoutez dans l'interface `ImageGrise` une méthode `ImageGrise optimiser()` qui retourne soit une instance de `ImageTab`, soit une instance de `ImageDict`, selon le contenu de l'image. Écrivez ensuite le code de ces méthodes dans les classes `ImageTab` et `ImageDict`.

Q 6. *QUESTION SUBSIDIAIRE :*
Lisez la documentation de `image.Viewer` et testez cet outil avec vos propres classes. **Qu'obtient-on si l'on ajoute un éléphant à un cheval ?**

Déposez les sources sur la plateforme moodle dans un fichier `monLogin-monBinome.tgz`
Pour créer un fichier `toto.tgz` dans le répertoire `travail` : `tar cvfz toto.tgz sources`