Examen Java

Décembre 2017

Durée : 3 heures

Instructions

- L'examen sera réalisé sur papier libre et avec **Eclipse**.
- Créer un nouveau projet dans **Eclipse** que l'on nommera **Exam1-<votre_nom>**.
- Pour chaque exercice, créer un nouveau package portant le nom de l'exercice en minuscule (e.g. **exercice1**) et y inclure toutes les classes créées pour répondre à l'exercice.
- Sauf instruction contraire, lorsque des vérifications ou des tests sont demandés dans un exercice, ne **pas** utiliser de `main` ...
- À la fin de l'examen, créer une archive (*e.g.* `.tgz`, `.zip`) du répertoire correspondant au projet **Exam1-<votre_nom>**
- Les exercices sont classés par ordre de difficulté croissante. Il est donc fortement conseillé de les traiter dans l'ordre.

Exercice 1

(Barème 30%)
On définit l'opération de normalisation comme étant l'application **affine** qui permet de passer d'un intervalle $[x_{\min}, x_{\max}]$ à l'intervalle $[0, 1]$:

$$\begin{aligned} f\colon [x_{\min}, x_{\max}] &\to [0, 1] \\ x &\mapsto \alpha x + \beta \end{aligned}$$

avec $f(x_{\min}) = 0$ et $f(x_{\max}) = 1$.

1. Ecrire l'équation de f, la fonction de normalisation (déterminer α et β).
2. Proposer une implémentation de f en tant que **fonction statique**.
   Indications :
   - Veiller à traiter les cas d'erreurs.
   - *Hint*

   ```
   public static double normalize(double x, double xmin, double xmax) throws Exception
   ```

3. Tester l'implémentation en considérant l'intervalle $[3, 6]$.
   - Que vaut $f(3)$? Ecrire le test correspondant.
   - Que vaut $f(6)$? Ecrire le test correspondant.
   - Que vaut $f(4.5)$? Ecrire le test correspondant.
   - Tester également les cas d'érreurs.
     hint :

     ```
     @Rule
     public ExpectedException expected = ExpectedException.none();

     @Test
     public final void test() throws Exception {
       //test code
       expected.expect(Exception.class);
       //more test code
     }
     ```

On souhaite généraliser la fonction de normalisation afin de transformer de manière affine les valeur d'un intervalle $[x_{\min}, x_{\max}]$ à un intervalle $[a, b]$.

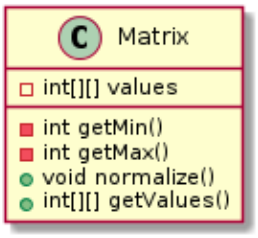
$$g\colon [x_{\min}, x_{\max}] \to [a, b]$$
$$x \mapsto ?$$

4. Écrire l'équation de g, la fonction *stretch*.
5. Proposer une implémentation de f en tant que **fonction statique**.
   Indications :
   - Veiller à traiter les cas d'erreurs.
   - *Hint*

   ```
   public static double stretch(double x, double xmin, double xmax, double a, double b) throws
       Exception
   ```

6. Tester l'implémentation en considérant les intervalles de départ $[3, 6]$ et d'arrivé $[0, 255]$.
   - Que vaut $f(3)$? Ecrire le test correspondant.
   - Que vaut $f(6)$? Ecrire le test correspondant.
   - Que vaut $f(4.5)$? Ecrire le test correspondant.
   - Tester également les cas d'érreurs.

La classe `Matrix` est définie par le diagramme ci-dessous :



- `getMin` permet de récupérer la valeur minimale du tableau `values`.
- `getMax` permet de récupérer la valeur maximale du tableau `values`.
- `normalize` permet d'*envoyer* les valeurs **courantes** d'un objet `Matrix` dans l'intervalle $[0, 255]$. Exemple :

$$\mathbb{A} = \begin{pmatrix} 50 & 0 & 100 \\ 0 & 50 & 0 \end{pmatrix} \qquad \text{normalize}(\mathbb{A}) = \begin{pmatrix} 127 & 0 & 255 \\ 0 & 127 & 0 \end{pmatrix}$$

7. Expliquer à quoi vont servir les méthodes `getMin` et `getMax`.
8. Implémenter complètement la classe `Matrix` avec les indications fournies.
   Indications : Ré-utiliser bien sûr l'implémentation précédente de la fonction `stretch`.
9. Tester l'implémentation.
   - En considérant la matrice de l'exemple $\mathbb{A}$, vérifier que $\mathbb{A}$.normalize() donne $\begin{pmatrix} 127 & 0 & 255 \\ 0 & 127 & 0 \end{pmatrix}$

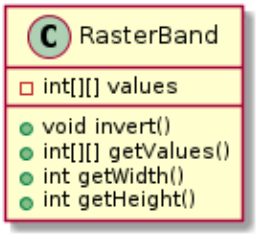
Exercice 2

(Barème 30%)

Configuration initiale

- Extraire le fichier `exercice2.zip` dans le répertoire `src` de votre projet **Eclipse**.
- Actualiser votre projet **Eclipse** en le sélectionnant dans **Eclipse** et en appuyant sur la touche *F5*, ou par un clic droit -> *Refresh*.

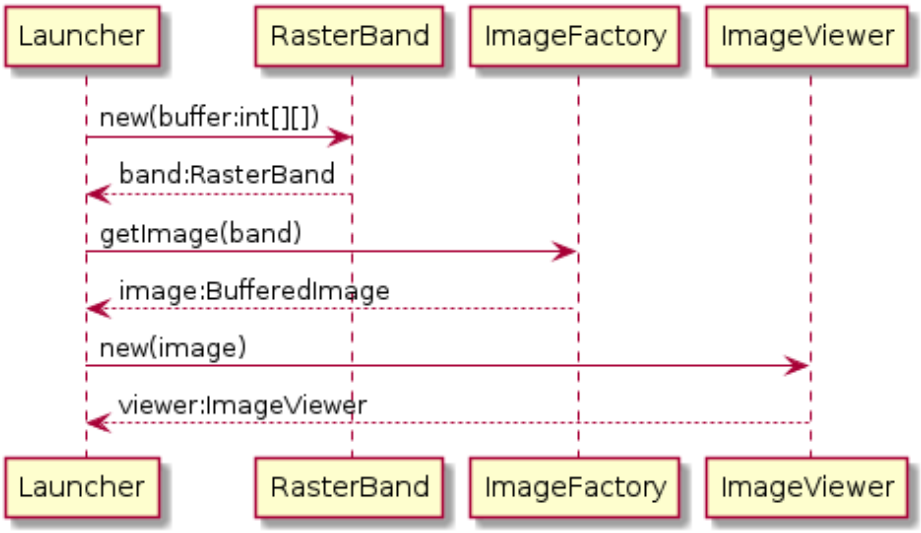
La classe `RasterBand` permet de modéliser une image 8 bits en niveaux de gris. Elle est définie par le diagramme ci-dessous :




```
/******** JAVADOC **************/
/**
* Returns the internal values.
*/
public void getValues();

/**
* Returns the number of columns in values.
*/
public void getWidth();

/**
* Returns the number of rows in values.
*/
public void getHeight();

/**
* Replace each element of values by its complement to 255.
* Example, given i,j, values[i][j] = 255-values[i][j]
*/
public void invert();
```

1. Implémenter complétement cette classe en vous aidant de la *Javadoc* fournie.

La classe `ImageViewer`, fournie dans l'archive `exercice2.zip` permet de visualiser dans une fenêtre graphique un objet `RasterBand`. Le diagramme de séquence ci-dessous décrit les étapes nécessaires pour créer un objet `ImageViewer` à partir d'un objet `RasterBand` :

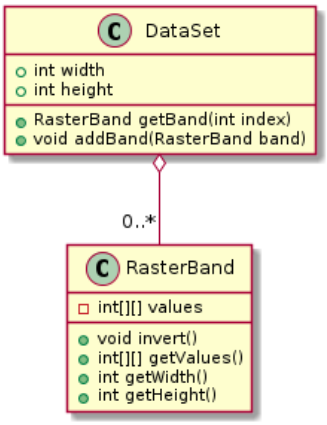


2. En utilisant le diagramme de séquence fourni, créer une classe de test `Launcher` avec une méthode `main` permettant de visualiser le buffer obtenu en appelant la méthode `Util.getLennaGreenBuffer()`.
   - Exécuter `Launcher` et vérifier qu'une fenêtre contenant une image apparait.
   - Que voyez-vous sur cette image ?
   - *Hint*

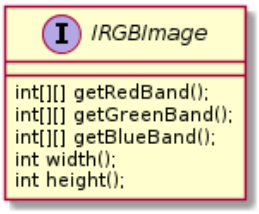
```
ImageViewer viewer; // Initialize with something sensible
viewer.pack();
viewer.setVisible(true);
```

3. Créer une deuxième classe de test `Launcher2` avec une méthode `main` permettant de visualiser un objet `RasterBand` créé avec le résultat de `Util.getLennaGreenBuffer()` et sur lequel on a appelé la méthode `invert()`.
   - Exécuter `Launcher2` et vérifier qu'une fenêtre contenant une image apparait.
   - Que voyez-vous sur cette image ? La comparer avec l'image précédente et la décrire.

La classe `DataSet` permet de modéliser une image 8 bits composée de plusieurs bandes chromatiques (e.g. rouge, vert, bleu, proche infra rouge...). Elle est définie par le diagramme ci-dessous :



Ainsi une image couleur RGB sera représenté par un objet `DataSet` ayant 3 objets `RasterBand` ordonnés comme suit : rouge, vert, bleu.
La classe `ImageFactory`, fournie dans l'archive `exercice2.zip` a une méthode `getRGBImage` permettant de créer une image couleur. Cette méthode prend en paramètre un objet de type `IRGBImage` décrit ci-dessous :



On souhaite pouvoir utiliser la méthode `getRGBImage` avec des objets de type `DataSet` **SANS** modifier le code de la méthode **ET** en faisant le moins de modifications possible au code existant.

4. Modifier/*Adapter* le code existant, tout en respectant les consignes données ci-dessus, pour utiliser la méthode `getRGBImage` avec des objets de type `DataSet`.
   - *Objectif bonus* Essayer de répondre à la question en ne modifiant **aucune** des classes existantes.
5. Créer une classe de test `Launcher3` avec une méthode `main` permettant de visualiser une image couleur avec les buffers fournis par la classe `Util`, i.e. `Util.getLennaRedBuffer(), Util.getLennaGreenBuffer(), Util.getLennaBlueBuffer()`.
6. De quelle couleur est le "boa" noué autour du chapeau ?

Exercice 3

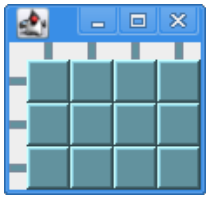
(Barème 40%)

Configuration initiale

- Extraire le fichier `exercice3.zip` dans le répertoire `src` de votre projet **Eclipse**.
- Actualiser votre projet **Eclipse** en le sélectionnant dans **Eclipse** et en appuyant sur la touche *F5*, ou par un clic droit -> *Refresh*.

Etudier attentivement le code fourni dans l'archive `exercice3.zip`. Ce code correspond à projet d'application graphique permettant de jouer à un picross dans une version à l'état de prototype.
On souhaite faire 2 évolutions au projet :

- On souhaite ajouter pour chaque ligne et pour chaque colonne un indicateur coloré, comme illustré par la figure ci-dessous :



  L'indicateur coloré a le comportement suivant :
  — Si toutes les cellules de la ligne (resp. colonne) sont de la même couleur, l'indicateur coloré prend la même couleur.
  — Si au moins deux cellules sont de couleurs différentes, l'indicateur coloré prend la couleur par défaut de l'arrière plan (e.g. gris).
- Actuellement, chaque fois que l'utilisateur clic sur une cellule, la cellule change d'état. Le changement d'état est cyclique, i.e. lorsqu'on atteint le dernier état, on revient sur le premier état. On souhaite ajouter un état supplémentaire, l'état `HYPOTHESIS` avec une couleur associée violet.

1. Dessiner le diagramme de classe correspondant au code fourni, en fournissant tout commentaire ou remarque que vous jugez utile à la compréhension du code fourni.
   - On pourra exécuter la classe "principale" `Launcher` et tester l'application pour aider à en comprendre le fonctionnement.
2. Objet *Cellule*
   - Quelles classes permettent de modéliser une *Cellule* ?
   - Quels sont les différents *états* dans lesquels une *Cellule* peut être ?
   - Expliquer ce qui se passe concrètement lorsqu'on clique sur une *Cellule*. Quelles sont les interactions entre les différentes classes mises en jeu (e.g méthodes appelées, impacts sur les objets. . .) ?
   - Les classes mises en oeuvre pour une *Cellule* utilise un *pattern* pour communiquer/intéragir. Lequel ?
3. Objet *Grille*
   - Expliquer simplement ce que représente la classe `Grid`.
   - Expliquer simplement ce que représente la classe `CellGroup`
   - Expliquer à quoi correspondent concrètement les attributs `lines` et `columns` de la classe `Grid`.

Evolution 1 : Ajout d'indicateurs Le composant graphique qui servira à représenter les indicateurs est déjà fourni, il s'agit de `LedWidget`

4. A quel classe va-t-on associer `LedWidget` pour répondre au besoin de l'évolution ?
5. Par quel mécanisme `LedWidget` va-t-il communiquer avec l'objet associé ?
6. Faire toutes les modifications nécessaires pour prendre en compte cette évolution.

Evolution 2 : Ajout d'un nouvel état

5. Dans quel type va-t-on définir le nouvel état ?
6. Dans quelle classe est conservé concrètement l'état d'une *Cellule* ?
7. Dans quel type va-t-on définir la nouvelle couleur ?
8. Quelle classe représente concrètement la couleur d'une *Cellule* ?
9. Faire **toutes** les modifications nécessaires pour prendre en compte ce nouvel état.