Examen Java

Décembre 2018

Durée : 3 heures

Instructions

(Coeff 1)

- L'examen sera réalisé avec **Eclipse** et **git**. Répondre aux questions libres sur copie.
- Utiliser
  - `JDK >= 1.8`
  - `JUnit 5`
  - `Eclipse Photon`
- **RESPECTER LES INSTRUCTIONS SUIVANTES À LA LETTRE**
- Le repository à utiliser est **https://github.com/CNAMPRO/18_stmn_java**.
- Travailler sur **votre** branche.
- Créer un nouveau projet **Eclipse CT** localisé sous `[racine_18_stmn_java]/Exam/CT`, `[racine_18_stmn_java]` étant le répertoire de votre clone git de **https://github.com/CNAMPRO/18_stmn_java**.
- 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.
- Lorsque des vérifications ou des tests sont demandés dans un exercice, ne **PAS** utiliser de `main` . . . , sauf instructions contraires.
- Ne **PAS** modifier la signature des fonctions/méthodes donnée dans les énoncés.
- À la fin de l'examen, faire un `commit+push` de tout le **code** correspondant à vos réponses.
  hint :

```
git add <files>
git commit
git push
```

- Vérifier que vos réponses sont disponibles dans votre branche sur **https://github.com/CNAMPRO/18_stmn_java**.

Set up initial

- Télécharger le fichier **https://github.com/CNAMPRO/18_stmn_java/blob/teacher/setup/alpha/setup.zip**
- Le décompresser à la racine de votre projet `CT`. Le répertoire `src` doit alors contenir les répertoires `exercie2` et `exercice3`.
- Actualiser votre projet **Eclipse** en le sélectionnant dans **Eclipse** et en appuyant sur la touche *F5*, ou par un clic droit -> *Refresh*.

Exercice 1

(Coeff 3)
On définit l'opération `transform` sur une matrice $\mathbb{A}$ **d'entiers** de n lignes et m colonnes comme suit :

$$\texttt{transform}(A_{i,j}) = B_{i,j}$$

avec

$$B_{i,j} = \begin{cases} \dfrac{A_{i-1,j} + A_{i,j-1} + A_{i,j} + A_{i,j+1} + A_{i+1,j}}{5}, & \text{pour } i = 1,\dots,n-2 \text{ et } j = 1,\dots,m-2 \\ A_{i,j}, & \text{sinon} \end{cases}$$

$$i = 0,\dots,n-1 \text{ et } j = 0,\dots,m-1$$

1. Implémenter la fonction `transform` (en tant que méthode statique d'une classe).
2. Tester l'implémentation avec la matrice suivante :

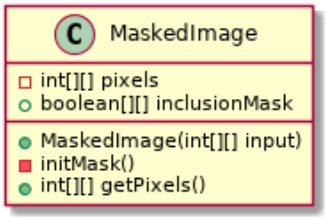
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 360 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

   hint : Quel est la matrice résultante attendue ?
3. *Hint*

```
public static final int[][] transform(int[][] a)
```

Exercice 2

(Coeff 6)
La classe `MaskedImage` permet de modéliser une image avec un masque, le masque permettant d'occulter une partie de l'image.
Le masque est un masque binaire de la même taille que l'image. `true` indique que l'on conserve la valeur de l'image, alors que `false` indique que l'on masque l'image.
La classe est définie par le diagramme ci-dessous :




```
/******** JAVADOC **************/
/**
* Construct a MaskedImage object.
* This constructor initializes:
* - pixels with the given input
* - inclusionMask by calling initMask
* @param input original image buffer used to initialize pixels
*/
public MaskedImage(int[][] input);

/**
* Initialize inclusionMask as an array of the same size
* as pixels and set all value to true.
*/
private void initMask();

/**
```

```java
* Compute pixel values by applying inclusionMask to pixels.
* Critical notice: this method does NOT modify attribute pixels.
* Let result the returned value. Then:
* - result[i][j] = pixels[i][j], if inclusionMask[i][j]=true
* - result[i][j] = 0, if inclusionMask[i][j]=false
* @return an array of the same size as pixels whose value is computed
* as described above.
*/
public int[][] getPixels();
```

1. Implémenter complétement cette classe en vous aidant de la *Javadoc* fournie.
2. Tester l'implémentation de `getPixels()` en considérant :
   - pixels =
   $$\begin{pmatrix} 2 & 2 & 2 \\ 4 & 4 & 4 \\ 6 & 6 & 6 \end{pmatrix}$$
   - inclusionMask =
   $$\begin{pmatrix} true & false & false \\ false & true & false \\ false & false & true \end{pmatrix}$$

On souhaite visualiser les images représentées par les objets `MaskedImage`. Pour cela, on utilise du code existant qui, pour des raisons externes, ne **DOIT PAS** être modifié. Il s'agit du code fourni dans le package `exercice2` par les classes `ImageViewer , ImageFactory, IImageBuffer`.

3. Etudier attentivement le code fourni et expliquer pourquoi il n'est pas possible de l'utiliser en l'état avec la classe `MaskedImage`.
4. Expliquer ce qu'il faudrait faire pour pouvoir utiliser le code fourni avec `MaskedImage`
5. Sans modifier les classes `ImageViewer, ImageFactory, IImageBuffer`, faire toutes les *adaptations* nécessaires pour utiliser `MaskedImage` avec le code fourni.
   - *Objectif bonus* Essayer de répondre à la question en ne modifiant **aucune** des classes existantes (i.e. même `MaskedImage`).

On souhaite visualiser l'image correspondant au *buffer* fourni par `Util.getLennaGreenBuffer()` en masquant le quartier bas gauche en créant un programme de test (avec un `main`).

6. Expliquer avec un diagramme de séquence comment réaliser un tel programme, que l'on nommera `Launcher`. On veillera à faire apparaître dans le diagramme toutes les classes et leurs méthodes mises en jeu pour réaliser le programme de test.

Partie optionnel/facultative/bonus
On souhaite visualiser l'effet de la fonction `transform` de l'Exercice1 sur une image.

6. Créer un programme de test que l'on nommera `Launcher2` permettant de visualiser l'image correspondant à `Util.getBoatBuffer()`
7. Ajouter la méthode `void transform()` à `MaskedImage` qui applique `transform` à `pixels`.
8. Créer un programme de test que l'on nommera `Launcher3` permettant de visualiser l'image correspondant à `Util.getBoatBuffer()` sur lequel a été appliqué `transform`.
9. Comparer les 2 images et en déduire à quoi correspond concrétement l'opération `transform`.

Exercice 3

(**COEFF 12**)

Etudier attentivement le code fourni dans le package `exercice3`.
Ce code correspond à un projet d'application graphique permettant de jouer à un démineur dans une version à l'état de prototype.

1. Tracer le diagramme de classe correspondant au code fourni, en fournissant tout commentaire ou remarque que vous jugez utile à la compréhension de la conception du code fourni.
   - On pourra exécuter la classe "principale" `Launcher` et tester l'application pour aider à en comprendre le fonctionnement.
2. Tracer le diagramme de séquence correspondant au click sur une case. On veillera à faire apparaître dans le diagramme toutes les classes et leurs méthodes mises en jeu pour traiter le click.

On souhaite réaliser les évolutions suivantes au projet :

(a) On souhaite que les nombres indiquant le nombre de bombes voisines à une case utilise le code couleur suivant :
   - 1 : bleu
   - 2 : vert
   - 3 : rouge
   - 4 : magenta

(b) On souhaite ajouter la possibilté d'utiliser des *flags* pour signaler la présence de bombe sur une case avec les caractéristiques suivantes
   - Le click droit permet de poser/enlever un flag sur une case **non déjà cliquée**.
   - Lors de la pose d'un flag, une icone representant le flag doit être affichée sur la case correspondante.
   - Un click gauche sur une case où un flag a été posé ne déclenche/révèle pas la case.

(c) Calcul automatique du nombre de bombes voisines pour chaque case où il n'y a pas une bombe. Actuellement le nombre de bombes voisines pour une case est renseigné manuellement à la création de la grille. On souhaiterait rendre le calcul automatique.

(d) Déclenchement en série. Lorsque l'on clique sur une case pour laquelle il n'y a pas de bombe voisine, cela révèle automatiquement toutes les cases voisines.

3. Réaliser l'évolution (a) couleur des nombres
   - Faire toutes les adaptations du code nécessaires
   - Ecrire une fiche de test permettant de valider l'évolution
   - *Hint*

```
protected Color getDisabledTextColor();
```

4. Réaliser l'évolution (b) flags
   - Faire toutes les adaptations du code nécessaires
   - Ecrire une fiche de test permettant de valider l'évolution
   - *Hint* : On pourra s'inspirer du code utilisé pour afficher l'icone des bombes et utiliser le fichier fourni `flag.jpg`
5. Réaliser l'évolution (c) calcul nombre de bombes
   - Faire toutes les adaptations du code nécessaires
   - Ecrire une fiche de test permettant de valider l'évolution
   - *Hint* : On pourra utiliser la méthode `getNeighboringCells()` de la classe `Grid`

```
class Grid {
  public void computeNeighboringBomb();
}
```

Partie optionnel/facultative/bonus

6. Réaliser l'évolution (d) Déclenchement en chaine