



## L3 informatique 2022-2023 : examen de contrôle continu

---

### Consignes :

- l'épreuve est individuelle et dure 2h ; les documents sont autorisés
- les déplacements et les discussions pendant l'épreuve sont interdits
- les téléphones portables sont interdits ; les ordinateurs portables sont autorisés
- L'accès à internet n'est autorisé que pour l'accès au moodle et au site de l'UE
- Les outils de communications (mail, chat, réseaux, etc.) sont interdits
- une archive contenant le code source à compléter est disponible sur le moodle de l'UE
- à la fin de l'épreuve, les fichiers java complétés devront être déposés sur le moodle
- seuls les fichiers déposés sur le moodle, et sans retard, seront corrigés

### Critères d'évaluation :

- réussite de la compilation et des tests
  - qualité de la solution algorithmique
  - qualité du code source (indentation, commentaires, choix des identifiants, etc.)
- 

### Intégration de l'archive dans un projet Java avec Eclipse :

- créer un projet Java (e.g. TPnote)
- faire un clic droit sur le dossier du projet (e.g. TPnote), *pas ailleurs*
- sélectionner Import... puis General>Archive File
- indiquer l'emplacement de l'archive téléchargée, puis cliquer sur Finish

### Organisation de l'archive :

- doc : documentation du code à compléter (cf. index.html)
- src/tp2022 : code à compléter et classe de tests
- tests : résultats attendus des tests

### Extraction sous Eclipse du code source complété à déposer sur Moodle :

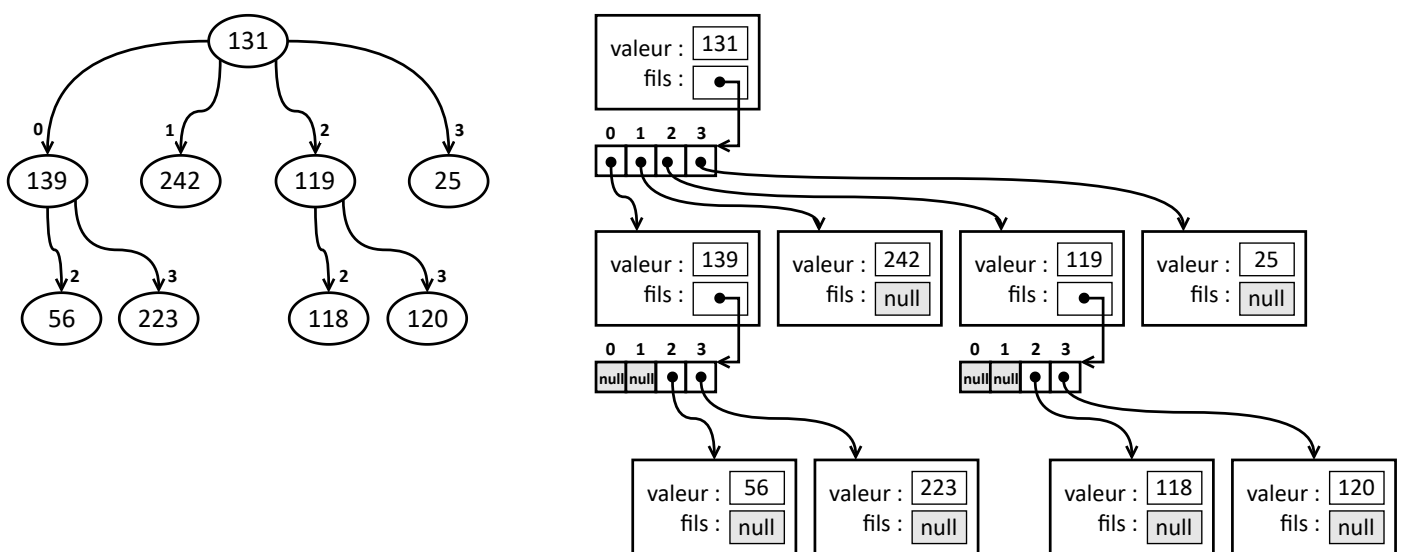
- faire un clic droit sur le package tp2022 (*pas ailleurs*)
  - sélectionner Export... puis General>File System
  - sélectionner les fichiers java à exporter
  - indiquer l'emplacement du répertoire cible, puis cliquer sur Finish
-

## Classe Quadtree

Un *quad tree* est un arbre dont chaque nœud stocke une valeur et dispose d'au plus 4 fils. On s'intéresse ici à la mise en œuvre de ce type d'arbre par la classe `Quadtree` qui stocke des valeurs entières et dont les fils sont désignés par des indices allant de 0 à 4 :

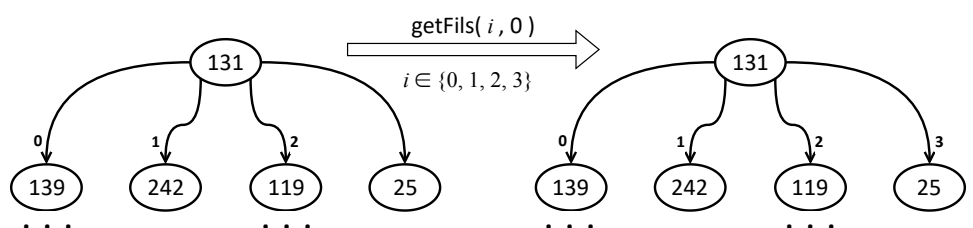
- attribut `valeur` (type `int`) : valeur stockée (obligatoirement comprise entre 0 et 255)
- attribut `fils` (type `Quadtree[]`) : référence `null` si l'arbre est réduit à une feuille (pas de fils), ou tableau de 4 cases contenant au moins une référence non `null` sinon

Dans l'illustration suivante, le *quad tree* a neuf nœuds. Il est représenté par neuf instances de `Quadtree` reliées entre elles. Notez bien les feuilles dont l'attribut `fils` est `null`, et les autres nœuds disposant d'un tableau de 4 cases, dont certaines stockent la référence `null` :

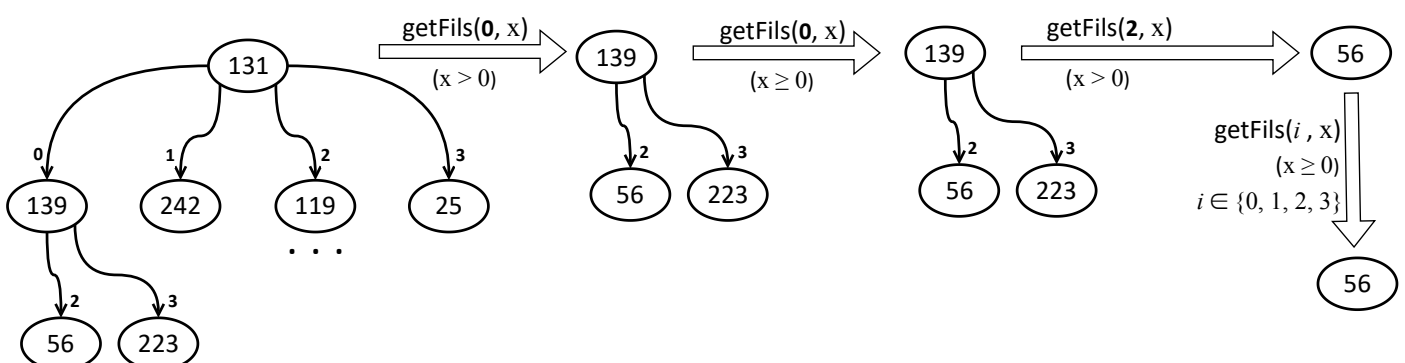


Dans un *quad tree*, la navigation vers un certain fils d'indice  $i$  est rendue possible par la méthode `getFils`, qui dépend de l'existence du fils  $i$ , ainsi que d'un certain *niveau de navigation* :

- si le niveau est 0 ou si le fils  $i$  n'existe pas, la navigation est stoppée : `getFils` renvoie `this` (on fait du sur-place). Illustration :



- sinon, le fils d'indice  $i$  demandé est renvoyé. Illustration :



## Représentation d'image

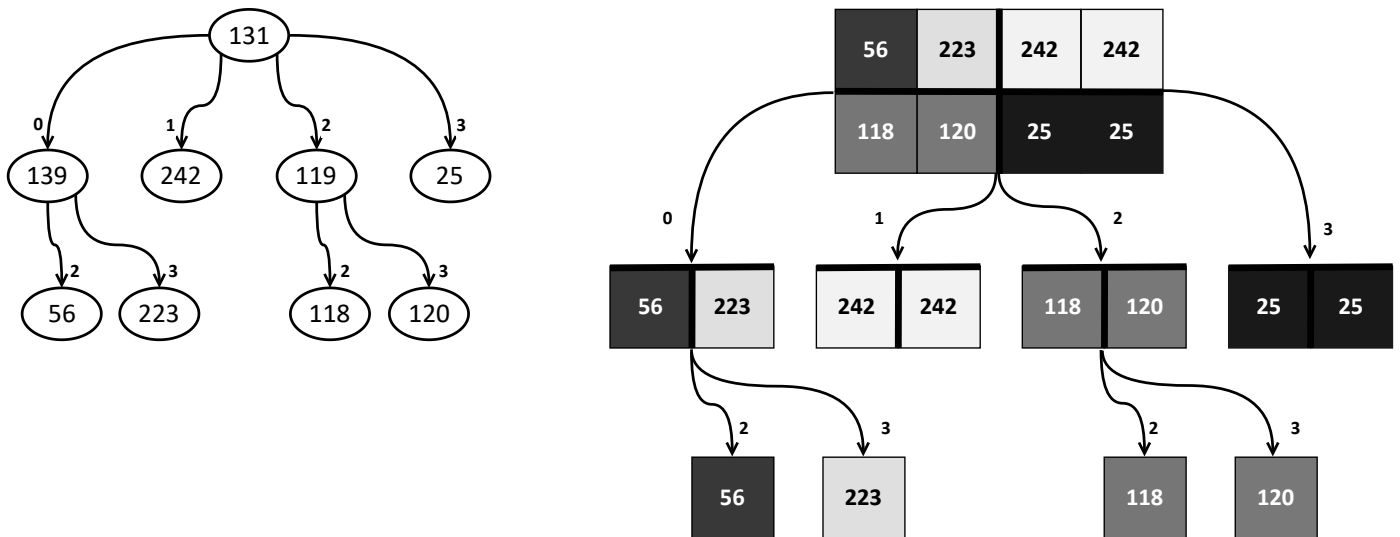
On utilise un *quad tree* pour représenter une image. Dans le cas particulier d'une image réduite à un seul pixel, le *quad tree* correspondant est réduit à une feuille et porte la valeur du pixel. Dans le cas général, les 4 fils d'un *quad tree* représentent 4 quarts de l'image :

- le fils 0 désigne le quart nord-ouest en haut à gauche
- le fils 1 désigne le quart nord-est en haut à droite
- le fils 2 désigne le quart sud-ouest en bas à gauche
- le fils 3 désigne le quart sud-est en bas à droite

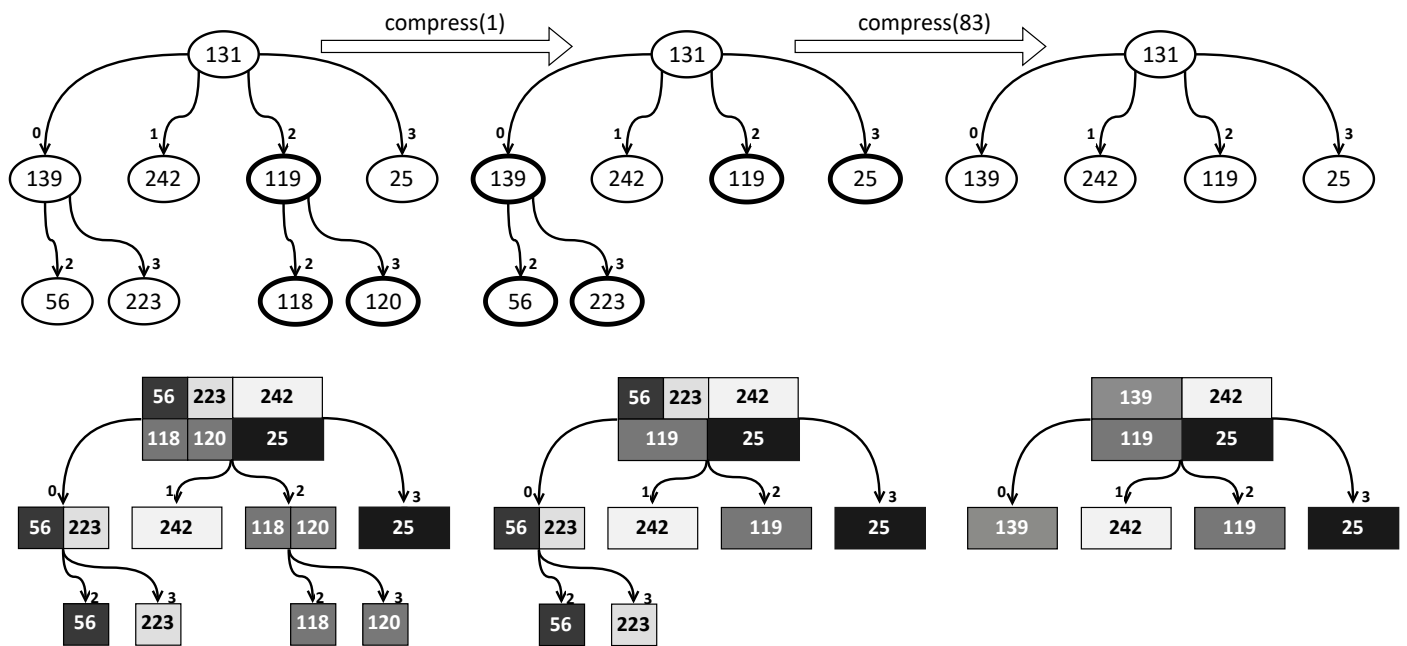
Dans ce cas général, la valeur stockée correspond à la moyenne des valeurs de pixels de l'image, *i.e.* la moyenne des moyennes stockées dans chaque quart.

Attention : puisque l'image n'est pas nécessairement carrée, un *quad tree* peut représenter une *ligne de pixels*, auquel cas il n'a que deux fils, typiquement d'indices 2 et 3 (sud) si la ligne est horizontale, ou d'indices 1 et 3 (est) si la ligne est verticale.

**Compression sans perte** Quand une image ne contient que des pixels de même valeur, il n'est pas nécessaire de les détailler : le *quad tree* correspondant est alors réduit à une feuille portant la valeur de ces pixels. Dans l'illustration suivante, une image de 4×2 pixels est représentée par un *quad tree*. Comme les quarts nord-est et sud-est ne contiennent que des pixels identiques, ils sont représentés par des feuilles :

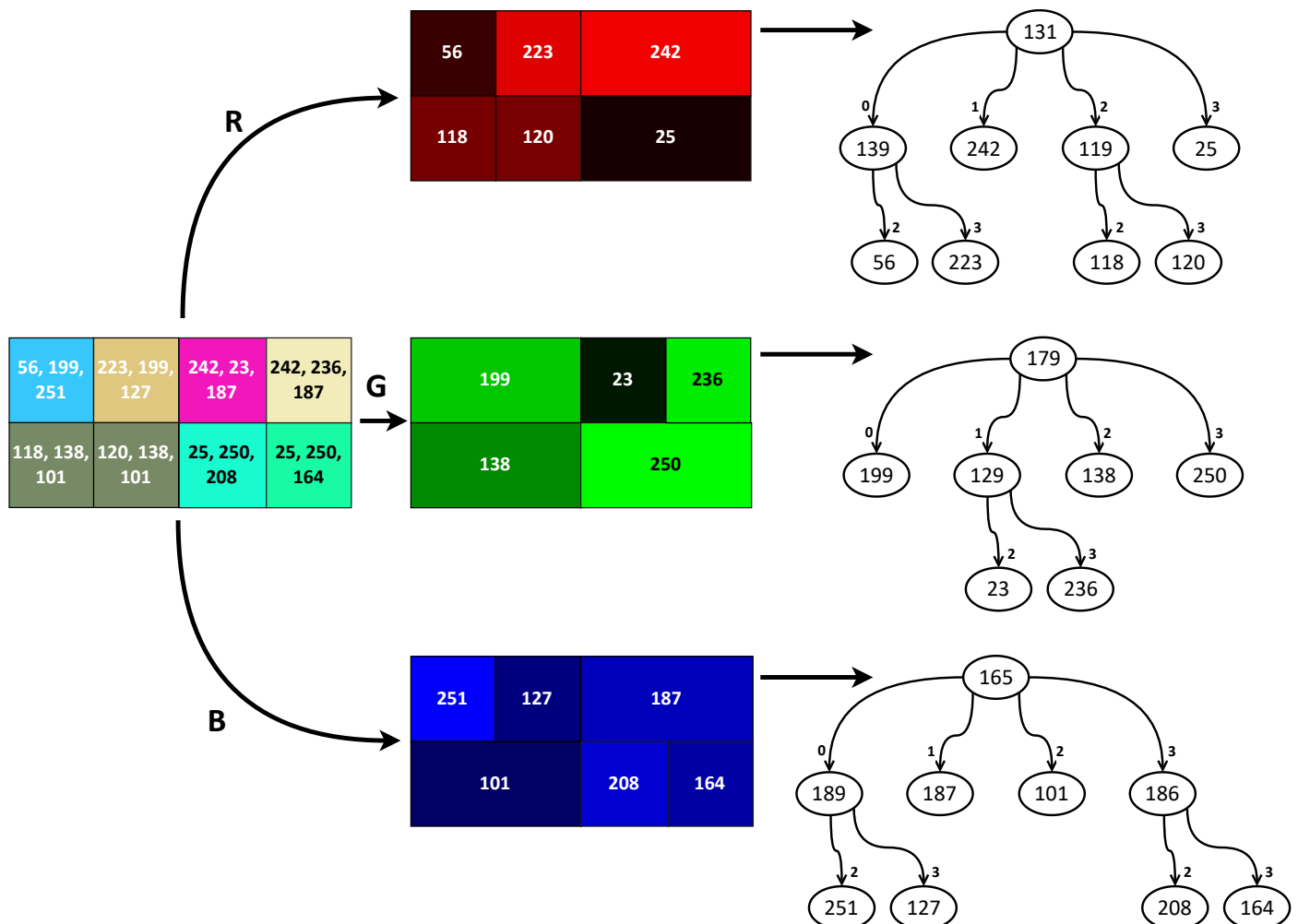


**Compression avec perte** Le principe de représentation des mêmes pixels par une seule feuille peut être étendu à la représentation de pixels *proches*, *i.e.* dont l'écart par rapport à leur moyenne ne dépasse pas une certaine valeur de tolérance. Dans l'exemple suivant, l'image est d'abord compressée avec une tolérance de 1, ce qui a pour effet d'assimiler les pixels de valeur 118 et 120 car leur écart par rapport à leur moyenne 119 ne dépasse pas 1. Dans un deuxième temps, l'image est compressée avec une tolérance de 83, ce qui a pour effet d'assimiler les pixels de valeur 56 et 223. L'image n'est pas globalement assimilée à 131 à cause du pixel 242 dont l'écart avec 131 dépasse la tolérance ( $242 - 131 = 111$ ) :

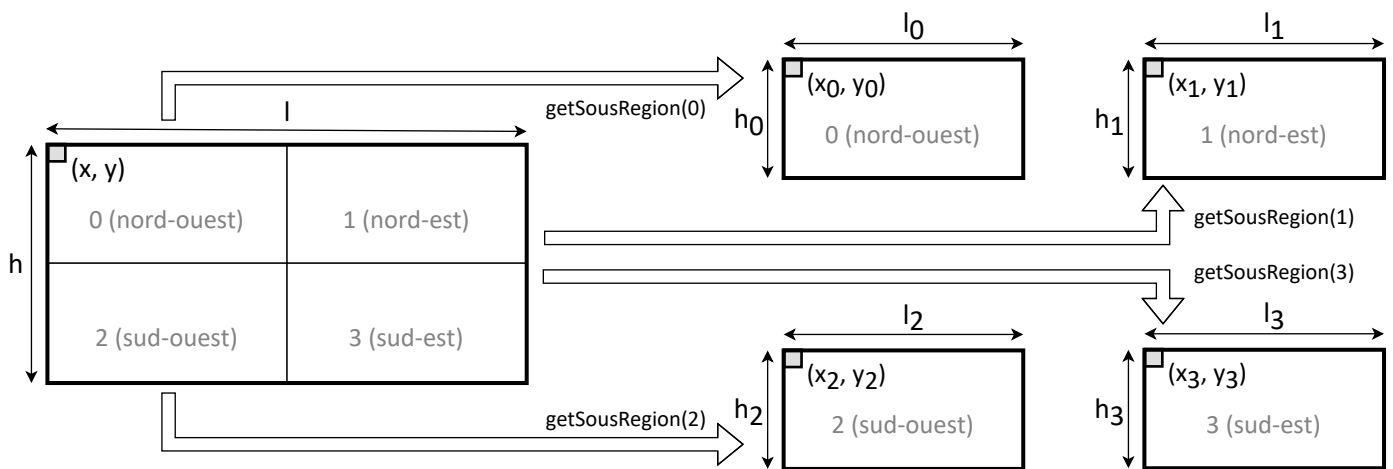


## Classes Image et Region

La classe Image représente une image par 3 *quad trees* : un par composante dans l'encodage RGB. Concrètement, chaque pixel d'une image correspond à la combinaison de valeurs d'intensité (comprises entre 0 et 255) des couleurs primaires rouge (R), vert (G) et bleu (B). Illustration :

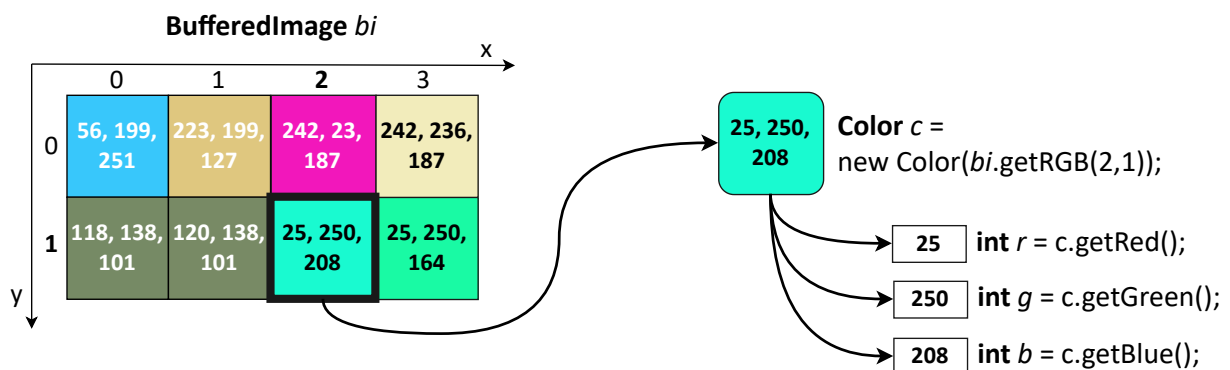


La classe **Région** représente une zone rectangulaire d'image par sa longueur, sa hauteur, et les coordonnées de son point en haut à gauche. La méthode `getSousRegion` permet de récupérer les quarts utilisés pour la construction des *quad trees* :

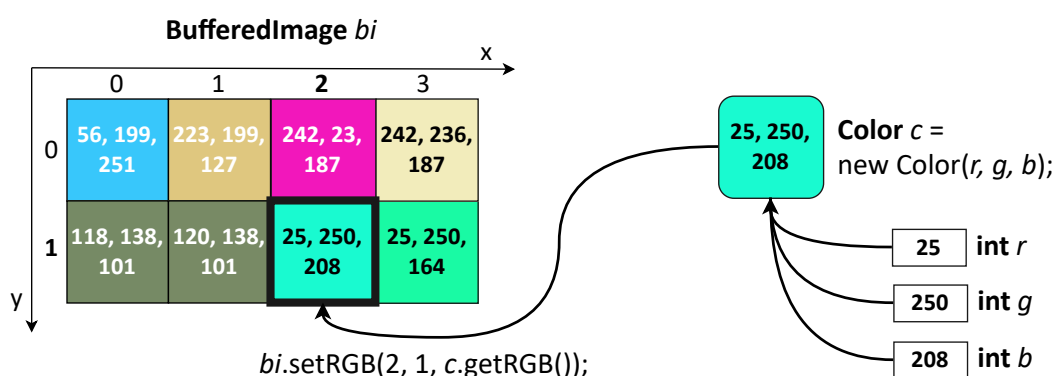


## Classes **BufferedImage** et **Color**

Les classes **BufferedImage** et **Color** sont fournies par l'API standard de Java. Elles représentent respectivement un tableau de pixels et une couleur dans l'encodage RGB. Pour récupérer les composantes RGB d'un pixel extrait d'un tableau, on utilise les méthodes `getRed`, `getGreen`, `getBlue`, `getRGB` et un constructeur de **Color** comme dans l'illustration suivante :

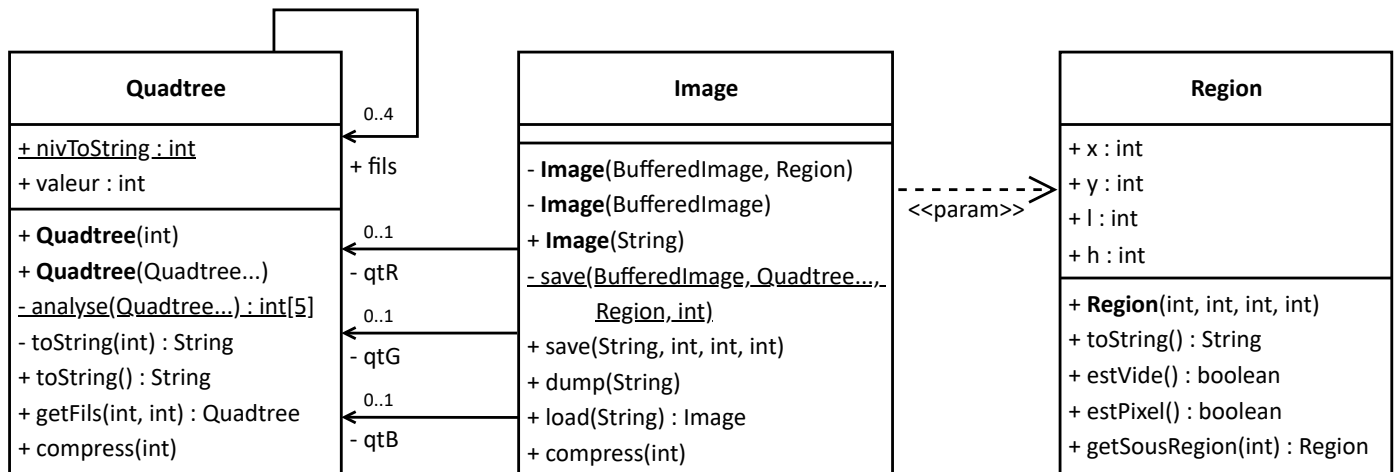


Inversement, pour écrire un pixel dans un tableau à partir de ses composantes RGB, on utilise les méthodes `setRGB`, `getRGB` et un autre constructeur de **Color** comme dans l'illustration suivante :



# Organisation du code

L'archive mise à disposition sur le Moodle contient une classe de test (Main), ainsi que les trois classes Quadtree, Region et Image à compléter :



## Travail à faire

Le code fourni doit être complété au maximum, dans la limite du temps imparti, de préférence en suivant les étapes suivantes, et en respectant les indications fournies en commentaire.

### Étape 1 : classe Quadtree

- ex 1 constructeur de feuille
- ex 2 constructeur de nœud à partir de ses fils et méthode auxiliaire analyse
- ex 3 méthode auxiliaire récursive de toString
- ex 4 méthode getFils
- ex 5 méthode compress

### Étape 2 : classe Region

- ex 1 constructeur
- ex 2 méthode toString
- ex 3 méthode estVide
- ex 4 méthode estPixel
- ex 5 méthode getSousRegion

### Étape 3 : classe Image

- ex 1 premier constructeur
- ex 2 méthode auxiliaire statique récursive save
- ex 3 méthode dump
- ex 4 méthode statique load
- ex 5 méthode compress