Aix*Marseille université
Initiative d'excellence
Année 2018-19

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TD 7
Code UE : SIN3U02

Images

1 Représentation d'image

On va considérer quatre manières de représenter une image en couleur et donc quatre classes d'images :

— `BruteRasterImage` : dans cette représentation, on stocke pour chaque pixel sa couleur sous la forme d'un objet `Color`. Une image de ce type sera construite à partir d'une matrice `Color[][] pixels`.

— `PaletteRasterImage` : dans cette représentation, on stocke :
  — une palette contenant au plus 128 `Color` qui correspondent aux couleurs utilisées dans l'image,
  — pour chaque pixel l'indice de sa couleur dans la palette avec un `byte`.
  Cette manière de faire a l'avantage d'économiser de la mémoire pour les images ayant peu de couleurs (un `byte` prenant beaucoup moins de place en mémoire qu'une `Color`). Une image de ce type sera construite à partir de `List<Color> palette` et `byte[][] indexesOfColors`.

— `SparseImage` : dans cette représentation, on stocke les coordonnées et la couleur de chaque pixel des pixels de l'image qui n'est pas blanc (`Color.WHITE`). Une image de type sera construite à partir de `List<Pixel> nonWhitePixels`. Pour cela, on a besoin d'une classe `Point` :

```
public class Point {
    public final int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

et d'une classe `Pixel` qui étend `Point` :

```
public class Pixel extends Point{
    private Color color;
    public Pixel(int x, int y, Color color) {
        super(x, y);
        this.color = color;
    }
    public Color getColor() {
        return color;
    }
}
```

— `VectorImage` : dans cette représentation, on stocke une liste de formes géométriques (utilisant interface `Shape` ci-dessous). Une image de type sera construite à partir de `List<Shape> shapes`. Chaque forme géométrique (qui peut être un cercle, un rectangle, ...) contient un certain nombre de points et définit la couleur des points à l'intérieur. La couleur d'un point de l'image est la couleur de la première forme dans la liste qui contient le point ou bien blanc si le point n'est contenu dans aucune forme.

```
public class Shape {
    /**
    * Tests if a specified Point is inside the boundary of the Shape.
    */
    boolean contains(Point point);
    /**
```

```
    * Return the color of the interior of the Shape.
    */
    Color getFillColor();
}
```

2 Exercices

1. On souhaite définir une interface `Image` qui sera implémentée par les 4 classes pouvant représenter une image. L'interface ne permet pas (pour le moment) de modifier l'image une fois que celle-ci est crée. D'après vous quelles peuvent être les méthodes définies par cette interface ?
2. Écrivez le code de la classe `BruteRasterImage` qui implémente l'interface `Image`. La classe devra avoir un constructeur prenant en paramètre une matrice `Color[][] pixels`.
3. En java, on peut définir un tableau de taille 0 et donc une matrice de couleurs de taille 0×0. On considère qu'une image ayant une hauteur ou largeur égale à zéro n'a pas de sens. Il faut donc lever une exception dans le cas où on nous donne une matrice dont la hauteur ou la largeur est 0. Rajouter la levée d'une exception de type `IllegalArgumentException` (exception existant déjà en Java) dans le constructeur correspondant à ce cas.
4. Une matrice en Java n'est pas forcément rectangulaire car chaque tableau représentant une "ligne" peut avoir une longueur différente. Lever une exception `IllegalArgumentException` dans le constructeur correspondant à ce cas.
5. Il existe maintenant deux cas qui entraînent la levée d'une exception de type `IllegalArgumentException`. Comment peut-on changer le code pour permettre à l'utilisateur de la classe `BruteRasterImage` de distinguer les deux cas possibles ?
6. Écrivez le code de la classe `PaletteRasterImage` qui implémente l'interface `Image`. La classe devra avoir un constructeur prenant en paramètre `List<Color> palette` et `byte[][] indexesOfColors`.
7. Quel est le code dupliqué entre les deux classes `BruteRasterImage` et `PaletteRasterImage` ?
8. Créez un classe abstraite `AbstractImage` qui sera étendue par `BruteRasterImage` et `PaletteRasterImage` afin d'éviter la duplication de code.
9. Écrivez le code de la classe `SparseImage` qui implémente l'interface `Image`. La classe devra avoir un constructeur prenant en paramètre `List<Pixel> nonWhitePixels`.
10. Écrivez le code de la classe `VectorImage` qui implémente l'interface `Image`. La classe devra avoir un constructeur prenant en paramètre `List<Shape> shapes`.
11. Afin de stocker les couleurs des pixels dans `SparseImage`, on peut utiliser l'interface `Map<K,V>`. Cette interface, qui est implémentée par `HashMap<K,V>`, permet d'associer des clés (de type `K`) à des valeurs (de type `V`). Elle contient entre autre les méthodes suivantes :
    - `boolean containsKey(Object key)` : Returns `true` if this map contains a mapping for the specified `key`.
    - `V get(Object key)` : Returns the value to which the specified `key` is mapped, or `null` if this map contains no mapping for the `key`.
    - `V getOrDefault(Object key, V defaultValue)` : Returns the value to which the specified `key` is mapped, or `defaultValue` if this map contains no mapping for the `key`.
    - `V put(K key, V value)` : Associates (maps) the specified `value` with the specified `key` in this map.

    Changer le code de la classe `SparseImage` pour utiliser un `Map` qui stockera les associations entre les positions et les couleurs des pixels.
12. On souhaiterait modifier les images après leur construction. Qu'est-ce qu'il faudrait modifier pour cela dans le code de `BruteRasterImage` et `PaletteRasterImage` ?
13. Même question pour les classes `SparseImage` et `VectorImage`.