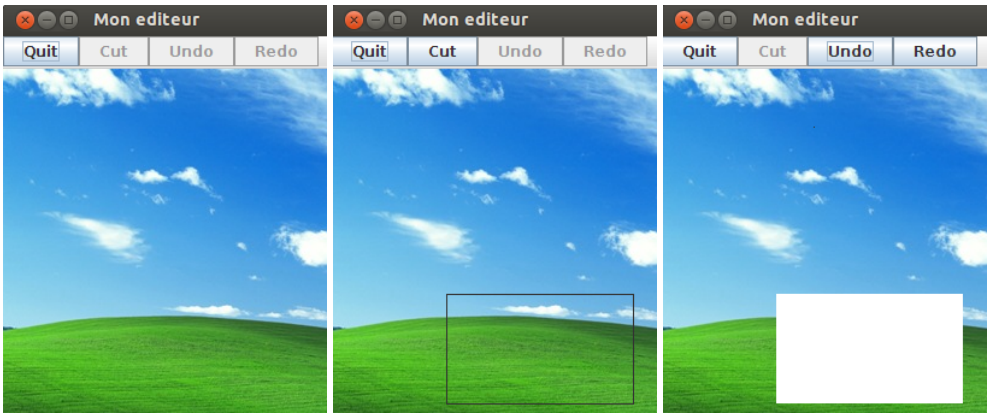
Université Paris Diderot **Programmation Orientée Objet**
L2-L3 **Année 2014-2015**

TP 9

Interfaces graphiques 3

N'hésitez pas à consulter la documentation Java habituelle :
– `https://docs.oracle.com/javase/7/docs/api/`
– `https://docs.oracle.com/javase/tutorial/uiswing/index.html`

Objectif de ce TP. Le but est de programmer un éditeur d'images. Nous nous contenterons ici d'une version très basique, qui permet :
– de charger une image du répertoire courant,
– de sélectionner une zone rectangulaire de l'image avec la souris,
– de couper la zone sélectionnée,
– d'annuler ou de refaire une action de coupe (undo/redo).



Exercice 1 - JFrame et JPanel

Commencez par créer une classe `ImageEdit` qui étend `JFrame`. Définissez une classe interne `ImagePane` qui étend `JPanel`. Ajoutez les champs suivants dans la classe `ImageEdit` :

– les JButton `quitButton`, `cutButton`, `undoButton` et `redoButton`
– un champ `ImagePane imagePane`
– un champ `UndoManager undoManager = new UndoManager();`

La classe `UndoManager`, présente dans `javax.swing.undo`, gère une liste de `UndoableEdits` et donne la possibilité d'annuler ou refaire des actions.

Dans la classe `ImagePane`, créer un champ `BufferedImage image`.

Exercice 2 - Constructeur d'ImageEdit

On crée ici le constructeur sans argument de la classe.

Commencez par définir un titre de votre fenêtre (avec `setTitle`), l'opération de fermeture (utiliser `setDefaultCloseOperation`), la barre de menu pour les boutons (utilisez `JMenuBar`, `setJMenuBar`). Instanciez les boutons avec leur label.

Rendez les boutons `undoButton`, `redoButton` et `cutButton` initialement non cliquables (utiliser `setEnabled`). Créez un `ActionListener` pour `quitButton` (quitter le programme) et `cutButton` (qui pour le moment ne fait rien).

Ajoutez les boutons à la barre de menus (utiliser `add`).

Enfin, instanciez le champ `imagePane` et définissez-le comme panneau courant (pour cela, utiliser `setContentPane`).

Exercice 3 - Constructeur d'ImagePane

Il s'agit de définir le constructeur sans arguments.

Chargez votre image favorite dans le champ image. Pour cela, on peut utiliser `ImageIO.read(new File(...))`, et il faut penser à gérer l'éventuelle exception levée. Vous pouvez définir une taille de fenetre avec la méthode `setPreferredSize(new Dimension( ... ))`.

Exercice 4 - La méthode paintComponent

Pour actualiser les affichages divers, nous allons redéfinir, dans la classe `ImagePane`, la méthode `paintComponent(Graphics g)` de JPanel comme suit :

```
  public void paintComponent(Graphics g) {

    super.paintComponent(g);

    g.drawImage(image, 0, 0, this);
}
```

Exercice 5 - Pour lancer l'application

Vous pouvez tester votre programme en écrivant la fonction `main` suivante dans la classe `ImageEdit` :

```
   public static void main(String [] args)  {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
          public void run() {
               ImageEdit view = new ImageEdit();
```

```
                view.pack();
                view.setVisible(true);
            }
        });
    }
```

Remarque : Cette manière de lancer l'application évite des problèmes d'affichage qui peuvent survenir dans certains cas.

Exercice 6 - Sélection d'une zone

Définissez la classe interne `Selection` dans la classe `ImagePane`. Cette classe doit étendre la classe `MouseAdapter` et implémenter `MouseMotionListener`. Les champs sont les coordonnées des deux points sélectionnés à la souris pour définir la zone rectangulaire. Vous devez ensuite implémenter les méthodes :

- `mousePressed`, qui met à jour les coordonnées de la zone en fonction du premier clic,
- `mouseDragged` qui met à jour les coordonnées du point d'arrivée,
- `mouseMoved` qui ne fait rien,
- `Rectangle getRectangle()`, qui renvoie les dimensions de la zone courante.

Créez un champ `Selection selection = new Selection()` dans la classe `ImagePane`.

Dans le constructeur d'`ImagePane`, fixez les contrôleurs d'actions de la souris (utiliser les méthodes `addMouseListener` et `addMouseMotionListener` sur le champ `selection`).

Enfin, pour actualiser l'affichage de sélection, ajoutez la ligne suivante à la méthode `paintComponent` : `((Graphics2D) g).draw(selection.getRectangle());` .

Exercice 7 - Lecture et écriture des pixels de l'image

On souhaite pouvoir manipuler les pixels de l'image. On définit la notion de coupe par la suppression d'une zone rectangulaire de pixels de l'image, qui consiste à remplir la zone avec des pixels blancs. On veut aussi pouvoir remplir une zone rectangulaire avec des pixels colorés.

Pour cela, créez dans la classe `ImagePane`, les méthodes suivantes :

- `void fillzone (Rectangle z, int[][] pixels)`, qui prend une zone et une matrice de pixels, puis remplit la zone associée de l'image (de type `BufferedImage`). On pourra utiliser tout d'abord deux méthodes de la classe `BufferedImage` :
  - `getRaster()` qui permet d'obtenir une version éditable de l'image sous forme d'un `WritableRaster`,
  - `getColorModel()` pour obtenir le modèle de couleur de l'image.

  Ensuite, pour chaque pixel de la zone, on obtient sa représentation dans le modèle de couleur de l'image via la méthode `getDataElements(int rgb, Object pixel)` de `ColorModel` : on choisit la valeur `null` pour `pixel`, et la valeur du pixel pour `rgb`. Enfin, utiliser ce résultat dans l'appel

à la méthode `setDataElements` de `WritableRaster` pour écrire le pixel dans l'image.

– `void clearzone (Rectangle z)`, qui remplit une zone par du blanc. Le principe ressemble à celui de `fillzone`. On utilise aussi `getRGB()` de la classe `Color`.

– `void clearzone ()`, qui appelle la méthode précédente sur le Rectangle obtenu à partir du champ `Selection`.

Exercice 8 - Zone de pixels mémorisée

Définissez la classe `Coupe`, interne à `ImageEdit`. Son rôle est de mémoriser une zone de pixels sous la forme d'une matrice, et d'agir dessus en fonction de la demande. Créez ses champs `Rectangle z` et `int pixels[][]`, puis son constructeur qui prend en arguments les coordonnées du coin de zone, ainsi qu'une `BufferedImage`, et remplit le champ `pixels` avec l'image. Vous pouvez utiliser la méthode `getRGB` dans `BufferedImage`, qui renvoie la valeur d'un pixel.

Enfin, implémentez ses méthodes

– `void doit()`, qui appelle la méthode `clearzone` sur `imagePane`,
– `void undo()`, qui appelle la méthode `fillzone` sur `imagePane`.

Exercice 9 - Coupes et UndoManager

La classe `UndoManager` manipule une liste d'objets implémentant l'interface `UndoableEdit`. Par ailleurs, la classe `AbstractUndoableEdit` est une implémentation abstraite de cette interface.

Définissez la classe `CutEdit` qui étend `AbstractUndoableEdit` et possède

– un champ `Coupe c`
– un constructeur `CutEdit(Coupe c)`
– une méthode `undo()` qui appelle sa méthode parente et la méthode `undo()` du champ
– une méthode `redo()` qui appelle sa méthode parente et la méthode `doit()` du champ.

Nous pourrons par la suite fournir directement des instances de `CutEdit` au `undomanager`.

Dans la classe `ImagePane`, ajouter les méthodes

– `CutEdit savecut(Rectangle z)` qui obtient la sous-image définie par `z` (utiliser la méthode `getSubimage`), l'utilise pour générer une nouvelle instance de `Coupe`, appelle `doit()` sur cette instance et renvoie une nouvelle instance `CutEdit(c)`,

– `CutEdit savecut()` qui appelle la méthode précédente sur le Rectangle obtenu à partir du champ `selection`.

Vous pouvez désormais implémenter le corps de la méthode `actionPerformed` pour `cutButton` dans le constructeur d'`ImageEdit` : il appelle `savecut()` sur `imagePane`, ajoute la modification à l'undo-

manager (utiliser `addEdit`), et rend cliquables les `undoButton` et `redoButton`.

Remarque sur les actions. Il arrive souvent que l'on souhaite pouvoir effectuer une même action via plusieurs composants différents. Par exemple, un bouton et un élément de menu peuvent tous deux actionner une sauvegarde de fichier. Swing fournit des interfaces et classes abstraites qui permettent de gérer aisément ce genre de situation.

L'interface `Action` est une base pour la création d'un objet implémentant une fonction utilisée par plusieurs composants. Cette interface étend l'interface `ActionListener` et contient donc la méthode `actionPerformed`. Ce type d'action contient un état courant indiquant si la fonctionnalité est activée ou non, des icônes...

En général, on étend la classe abstraite `AbstractAction` qui implémente `Action` et l'on redéfinit `actionPerformed`. Ensuite, on attache l'action aux divers composants.

Exercice 10 - Actions undo et redo

Nous allons utiliser `AbstractAction` pour nos actions undo et redo. Dans la classe `ImageEdit`, définissez les classes internes :

- `UndoAction` qui étend `AbstractAction`, appelle la méthode `undo()` du manager quand c'est possible (tester avec `canUndo()`),
- `RedoAction` qui étend `AbstractAction`, appelle la méthode `redo()` du manager quand c'est possible (tester avec `canRedo()`).

Ajouter les champs `UndoAction undoAction` et `RedoAction redoAction` à la classe `ImageEdit`. Initialisez ces champs dans le constructeur d'`ImageEdit` et modifiez la création de vos JButton (par exemple, `undoButton = new JButton(undoAction);`).

Pour le moment, nous n'utilisons pas tout le potentiel d'`AbstractAction`, mais cela pourra être fait par la suite à partir du squelette que nous venons d'établir.

Exercice 11 - Dernières vérifications

Vérifiez que vos classes et méthodes sont correctement agencées, et complétez les éventuels trous dans votre programme : initialisations, etc.

Vous devriez maintenant pouvoir éditer l'image, en sélectionnant et supprimant des zones, avec possibilité d'annuler vos actions ou de les exécuter à nouveau.

Exercice 12 - Exercice bonus

En guise d'extension, vous pouvez ajouter la possibilité de copier une zone de l'image (via une sélection préalable) et de la coller dans une autre zone (via, par exemple, une autre sélection ou le positionnement du coin par un clic). Vous pouvez aussi étudier la possibilité d'un drag-and-drop, ajouter des icones pour vos boutons (utiliser le potentiel d'AbstractAction) et utiliser les touches du clavier et un menu déroulant pour effectuer les mêmes actions (nouvelle utilisation d'AbstractAction).