Université Paris Cité **POOIG**
L2 **Année 2022-2023**

TP n°10

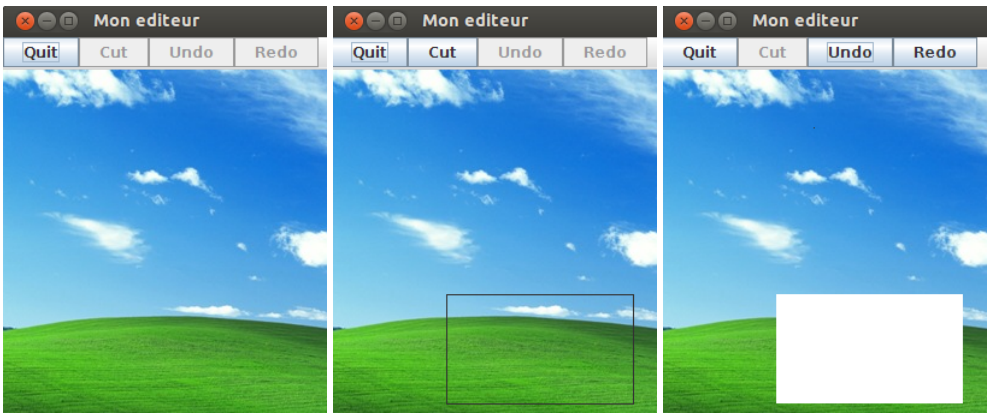
Interfaces graphiques 2

Dans ce TP, il est essentiel de consulter fréquemment la documentation Java habituelle sur les différentes classes prédéfinies qu'on utilise :

— `https://docs.oracle.com/en/java/javase/11/docs/api/index.html`
— `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).



1 Premières fonctionnalités

Exercice 1 – Le Modèle

Commencez par créer une classe `ImageEditModel`. Pour le moment cette classe ne contient qu'un champ `BufferedImage image`. Écrivez un constructeur de la classe `ImageEditModel` qui prend en paramètre un string représentant le chemin de l'image que vous allez charger dans le champ `image`. Pour cela, on peut utiliser l'instruction :

```
ImageIO.read(new File(chemin));
```

Pensez à gérer l'éventuelle exception levée.

Écrivez également une méthode `getImage()` qui retourne le champ `image`.

Exercice 2 – Écrire sur l'image

1. Écrivez une méthode `public void fillzone(Rectangle z, int[][] pixels)` dans la classe `ImageEditModel` qui modifie la zone de `image` correspondant au rectangle z. On va remplacer la couleur de chaque pixel dans cette zone par la valeur indiqué dans la matrice `pixels`. La taille de la matrice doit donc correspondre à la taille du rectangle. Pour changer la couleur d'un pixel, on peut utiliser la méthode `setRGB(...)` de la classe `BufferedImage` (pensez à consulter la documentation).
2. Écrivez une méthode `public void clearzone(Rectangle z)` dans la classe `ImageEditModel` qui va aller mettre des pixels blancs dans la zone de l'image représentée par le rectangle z. Pour récupérer l'entier représentant le code sRGB de la couleur blanc, on pourra utiliser les intructions suivantes :
   ```
   Color color = Color.white;
   int srgb = color.getRGB();
   ```
   Vour pourriez faire appel à la méthode `fillzone` à l'intérieur de `clearzone`.

Exercice 3 – La classe `ImageEditView`

Créez maintenant une classe `ImageEditView` qui étend la classe `JFrame`. Définissez une classe interne `ImagePane` qui étend la classe `JPanel`.

Ajoutez les champs suivants dans la classe `ImageEditView` :
- Les JButton `cutButton`, `undoButton` et `redoButton`,
- Un champ `ImagePane imagePane`,
- Un champ `ImageEditModel model`.

Exercice 4 – La classe interne `ImagePane`

1. Dans la classe `ImagePane` ajoutez le champ `Selection selection = new Selection();`
   `Selection` est une classe interne que l'on définira plus tard.
   Pour actualiser les affichages divers, nous allons redéfinir (dans la classe `ImagePane`) la méthode `paintComponent(Graphics g)` héritée de la classe `JPanel` comme suit :
   ```
   public void paintComponent(Graphics g) {
     super.paintComponent(g);
     g.drawImage(model.getImage(), 0, 0, this);
   }
   ```
2. Écrivez le constructeur par défaut de ImagePane. Dans celui-ci,
   - vous définirez la taille du panneau `ImagePane` avec la méthode `setPreferredSize(new Dimension(...))`. On veut que la taille du panneau correspond à la taille de l'image. Regardez la documentation pour comprendre comment utiliser `Dimension`, et comment récupérer la taille de l'image.
   - vous enregistrerez les contrôleurs d'action de la souris :
     ```
     addMouseListener(selection);
     addMouseMotionListener(selection);
     ```
     Cela exige que l'objet `selection` implémente les méthodes `mousePressed`, `mouseDragged`, `mouseMoved`, qu'on va implémenter plus tard.

Exercice 5 – Sélectionner une zone de l'image

Définissez maintenant la classe interne `Selection` dans la classe `ImagePane`. Cette classe doit étendre la classe `MouseAdapter` et implémenter `MouseMotionListener`. Elle possède quatre champs de type `int` représentant les coordonnées des deux points sélectionnés pour définir la zone rectangulaire. Vous devez ensuite implémenter les méthodes :

- `Rectangle getRectangle()`, qui renvoie un rectangle représentant la zone sélectionnée (on pourra consulter la documentation pour trouver le constructeur approprié de la classe `Rectangle`).
- `void mousePressed(MouseEvent event)` qui met à jour les coordonnées du point de départ, puis désactive le bouton `cutButton` (utilisez `setEnabled`) et actualise l'affichage de `ImagePane` (utilisez l'instruction `repaint();`).
- `void mouseDragged(MouseEvent event)` qui met à jour les coordonnées du point d'arrivée, active le bouton `cutButton` (si jamais le point d'arrivée est différent du point de départ) et actualise l'affichage de `ImagePane`.
- `void mouseMoved(MouseEvent event)` qui ne fait rien.

Regarder la documentation de la classe `MouseEvent` pour comprendre comment accèder à la position de la souris.

Enfin, afin d'actualiser l'affichage de la sélection, ajoutez la ligne

```
((Graphics2D) g).draw(selection.getRectangle());
```

à la méthode `paintComponent` de la classe `ImagePane`.

Observez qu'on n'a pas écrit un constructeur dans la classe `Selection`. C'est parce que le constructeur par défaut nous suffit : les champs entiers pour les coordonnées sont initialisés à l'intérieur de chaque méthode avant d'être utilisés.

Exercice 6 – Le constructeur de `ImageEditView`

On se propose maintenant d'implémenter un constructeur de la classe `ImageEditView` qui prend en paramètre un objet de type `ImageEditModel`.

1. Instanciez le champ `model` de la classe courante en utilisant le paramètre du constructeur.
2. Définissez le titre de votre fenêtre (avec `setTitle`), l'opération de fermeture (utilisez la méthode `setDefaultCloseOperation`), la barre de menu pour les boutons (utilisez `JMenuBar` et `setJMenuBar`). Instanciez les boutons avec leur label.
3. Rendez les boutons initialement non-cliquables (utilisez `setEnabled`) et ajoutez les boutons à la barre de menus (utlisez `add`).
4. Instanciez le champ `imagePane` et associez-le au `ContentPane` de la fenêtre (utilisez `setContentPane`).

Re-regardez le TP9 pour cela si besoin.

Exercice 7 – Pour lancer l'application

Créez une classe `Launcher` qui contient un `main` permettant de lancer votre programme graphique. Dans ce main, il faut d'abord créer un objet de type `ImageEditModel` à partir d'une image qui se trouve dans votre répertoire, puis créer un objet de type `ImageEditView` avec en paramètre l'objet de type `ImageEditModel` que vous venez de créer.

Dimensionnez la fenêtre en fonction de ses composants (utilisez la méthode `pack()` de votre objet de type `ImageEditView`) et rendez-la visible (utilisez `setVisible(true)`).

On rappelle que pour être "thread-safe" il faut utiliser l'invocation `EventQueue.invokeLater(...)` que nous avons vue dans le cours.

Votre programme devrait maintenant ouvrir une fenêtre qui affiche l'image que vous avez choisie et sur laquelle vous pouvez sélectionner un rectangle (mais les boutons ne font rien pour le moment).

2 Undo et Redo

Exercice 8 – La classe `Coupe`

1. Définissez une classe `Coupe`, interne à `ImageEditModel`. 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 argument les coordonnées du coin supérieur gauche de la zone, la largeur et la hauteur de la zone, ainsi qu'une `BufferedImage`. Le constructeur remplit le champ `pixels` avec les valeurs RGB des pixels de l'image. Vous pouvez utiliser la méthode `getRGB` de la classe `BufferedImage`, qui renvoie la valeur d'un pixel.
2. Implémentez ses méthodes
   — `void doit()`, qui appelle la méthode `clearzone` sur l'objet courant de la classe englobante.
   — `void undo()`, qui appelle la méthode `fillzone` sur l'objet courant de la classe englobante.

Exercice 9 – La classe `CutEdit`

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

1. Définissez une classe `CutEdit`, interne à `ImageEditModel` qui étend `AbstractUndoableEdit`. Elle contient un unique champ `Coupe c`.
2. Écrivez le constructeur de cette classe qui prend en argument un objet de type `Coupe`.
3. Implémentez les méthodes `undo()` et `redo()` héritées de `AbstractUndoableEdit` qui appellent (respectivement) les méthodes `undo()` et `doit()` de l'object `Coupe c`.

Exercice 10 – Save and Cut

1. Ajoutez un champ `UndoManager undoManager = new UndoManager()` à la classe `ImageEditModel`.
2. Écrivez une méthode `public void saveCut(Rectangle z)` dans la classe `ImageEditModel` qui crée un objet coupe correspondant au rectangle `z`, effectue la coupe sur l'image et sauvegarde l'opération dans `undoManager`. De manière détaillée, cette méthode doit :
   — Récupérer la sous-image du champ `image` correspondant à la zone du rectangle `z` (utilisez la méthode `getSubimage` de la classe `BufferedImage`).
   — Créer un objet `Coupe c` à partir des coordonnées du coin en haut à gauche de `z` et de la sous-image récupée précédemment.
   — Effectuer la coupe de la zone (en appelant la méthode `doit()` de `c`).
   — Créer un nouvel objet de type `CutEdit` avec `c` en paramètre et ajouter cet objet à `undoManager` (utilisez la méthode `addEdit` de la classe `UndoManager`).

On va maintenant enfin pouvoir implémenter les actions correspondant aux boutons "Cut", "Undo" et "Redo" de la fenêtre.

Exercice 11 – Boutons Cut, Undo et Redo

On rappelle que pour effectuer une action en fonction d'un évènement graphique (par exemple le "cliquage d'un bouton"), il faut définir une classe (interne) qui implémente `ActionListener` et il faut enregistrer cet `ActionListener` auprès le composant qui va potentiellement générer un évènement (par exemple un bouton), en utilisant `addActionListener(...)`. Notons que la classe qui implémente `ActionListener` peut être anonyme ou même être définie avec une expression lambda comme vu en cours. En utilisant cette dernière option pour le bouton "Cut", votre code de la première question ci-dessous devrait ressembler à ça :

```
cutButton.addActionListener(
  (ActionEvent e) -> {
    // instructions
  });
```

1. Dans le constructeur de la classe `ImageEditView`, ajoutez un `ActionListener` au bouton "Cut". Celui-ci devra :
   — Effectuer l'opération `saveCut` sur `model` avec en paramètre le rectangle récupéré dans le champ `selection` de `imagePane`.
   — Actualiser l'affichage de `imagePane` (utilisez `repaint()`).
   — Désactiver le bouton "Cut" et activez les boutons "Undo" et "Redo".
2. Dans le constructeur de la classe `ImageEditView`, ajoutez un `ActionListener` au bouton "Undo". Celui-ci devra appeler la méthode `undo()` du champ `undoManager` de `model` puis actualiser l'affichage de `imagePane`. Notez qu'avant d'effectuer l'opération de "undo" il faut vérifier que cette opération est possible (utilisez `canUndo()` de la classe `UndoManager`).
3. Même chose que la question précédente avec le bouton "redo" et l'opération "redo()".

Votre programme devrait maintenant marcher avec les fonctionnalités voulues.