TD 3 : Gestion des événements

Objectifs

- se familiariser avec les différentes façons de gérer les événements,
- mise en pratique des listeners.

Exercice 1. Mise en application des listeners

La Figure 1 présente une capture d'écran d'une interface avec un potentiomètre et un champ de texte. La manipulation du potentiomètre doit mettre à jour la valeur correspondante affichée dans le champ de texte. Réciproquement, la saisie d'un nombre dans le champ de texte doit mettre à jour la position du potentiomètre.

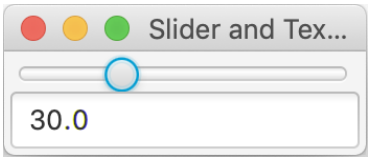

FIGURE 1 – Capture d'écran de l'interface avec potentiomètre et champ de texte.

Q 1. Le modèle associé à un potentiomètre est représenté par la classe DoubleProperty [1], qui s'obtient avec la méthode `valueProperty` de la classe `Slider`. Donnez le code pour s'abonner et être notifié des changements de position du potentiomètre et mettre à jour le champ de texte.

Q 2. Le modèle associé à un champ de texte est représenté par la classe StringProperty [2], qui s'obtient avec la méthode `textProperty` de la classe `TextField`. Donnez le code pour s'abonner et être notifié des changements de valeur du champ de texte et mettre à jour le potentiomètre.

Exercice 2. Logiciel de dessin

Dans le cadre de la réalisation d'un logiciel de dessin, nous souhaitons permettre la configuration de la taille et de la forme de « pinceaux ». Nous proposons dans un premier temps l'interface de configuration représentée sur la Figure 2.

Il doit être possible de modifier la taille du pinceau de plusieurs façons :

- en déplaçant le potentiomètre,
- en entrant une valeur numérique dans le champ de texte,
- en cliquant directement sur l'image représentant la forme : la taille est ajustée suivant la distance entre le pointeur souris et le centre de la forme.

De plus, il doit être possible d'alterner entre une forme circulaire et une forme carrée à l'aide de boutons radio. Le code suivant permet l'affichage et le positionnement de tous les éléments tels qu'ils apparaissent sur la Figure 2 mais la gestion des événements est incomplète. Il vous est demandé, dans les questions de cet exercice, de compléter le code. Pour chaque question, vous donnerez les modifications et ajouts nécessaires en précisant les numéros de lignes concernés.

Q 1. La gestion du potentiomètre se fait par l'intermédiaire d'une classe interne implémentée dans la classe *PencilConfig*. La taille du pinceau s'adapte correctement, mais pas la valeur du champ de texte,

1. https://docs.oracle.com/javase/8/javafx/api/javafx/beans/property/DoubleProperty.html
2. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextInputControl.html#textProperty--

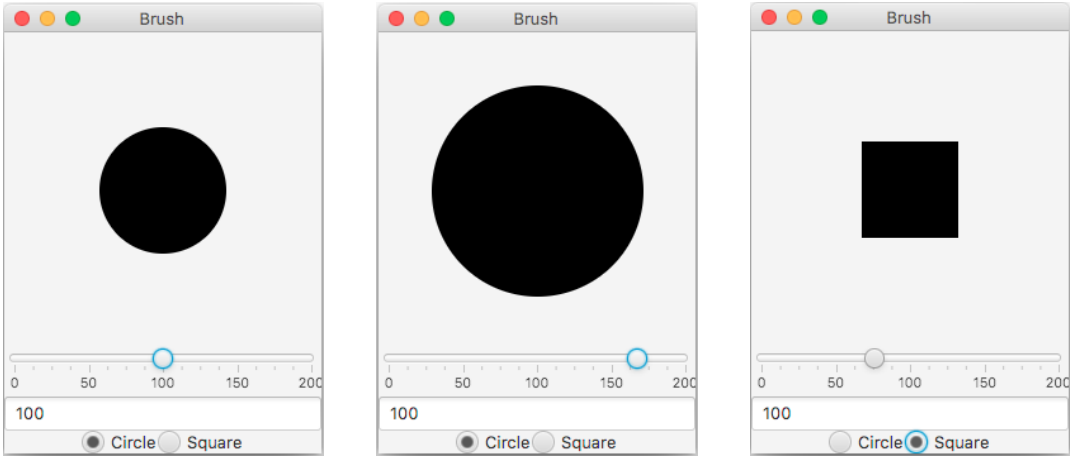

FIGURE 2 – Captures d'écran de l'interface. (gauche) : configuration initiale ; (milieu) ajustement du potentiomètre augmentant la taille du pinceau, mais ne modifiant pas la valeur indiquée dans le champ de texte ; (droite) modification de la forme du pinceau à l'aide des boutons radio.

qui reste à sa valeur initiale, comme illustré avec la figure 2 (milieu). Quel est le problème ? Proposez une correction.

Q 2. Nous souhaitons maintenant que l'utilisateur puisse modifier la valeur directement dans le champ de texte, et que cette modification soit répercutée directement sur le potentiomètre ainsi que sur la représentation du pinceau. Proposez une solution directement dans la classe principale *PencilConfig*, en implémentant une interface de gestion d'événements.

Q 3. Lorsque l'utilisateur clique sur la représentation du pinceau, la taille du pinceau doit atteindre la position du pointeur, c'est-à-dire qu'elle doit prendre une valeur égale à deux fois la distance entre le centre du canvas et la position du clic du pointeur. Proposez une solution en utilisant une nouvelle classe interne.

Q 4. Enfin, nous souhaitons que la forme choisie change en modifiant l'attribut booléen *circle* grâce aux boutons radio. Proposez une solution en utilisant des lambda expressions. Indication : ajoutez un listener sur *group.selectedToggleProperty()* pour savoir quel bouton radio a été sélectionné.

```
1  public class PencilConfig extends Application {
2
3    class SliderEvent implements ChangeListener<Number> {
4      public void changed(ObservableValue<? extends Number> observable,
5          Number oldValue, Number newValue) {
6        size = newValue.intValue();
7        repaintCanvas();
8        field = new TextField(Integer.toString(size));
9      }
10   }
11
12   private Canvas canvas;
13   private Slider slider;
14   private TextField field;
15   private boolean circle = true;
16   private int size = 100;
17
18   public void start(Stage stage) {
19
20     canvas = new Canvas(250, 250);
21
22     slider = new Slider();
23     slider.setMin(0);
24     slider.setMax(200);
25     slider.setValue(100);
```

```
    slider.setShowTickLabels(true);
    slider.setShowTickMarks(true);
    slider.setMajorTickUnit(50);
    slider.valueProperty().addListener(new SliderEvent());

    field = new TextField(Integer.toString(size));
    field.setMaxSize(Integer.MAX_VALUE, Integer.MAX_VALUE);

    ToggleGroup group = new ToggleGroup();
    RadioButton rb1 = new RadioButton("Circle");
    rb1.setToggleGroup(group);
    rb1.setSelected(true);
    RadioButton rb2 = new RadioButton("Square");
    rb2.setToggleGroup(group);

    HBox toggleBox = new HBox();
    toggleBox.setAlignment(Pos.CENTER);
    toggleBox.getChildren().add(rb1);
    toggleBox.getChildren().add(rb2);

    VBox root = new VBox();
    root.getChildren().add(canvas);
    root.getChildren().add(slider);
    root.getChildren().add(field);
    root.getChildren().add(toggleBox);

    Scene scene = new Scene(root);
    stage.setTitle(" Brush ");
    stage.setScene(scene);
    stage.show();
    this.repaintCanvas();
  }

  private void repaintCanvas() {
    GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.clearRect(0, 0, 250, 250);
    int position = 125 - size / 2;
    if (circle)
      gc.fillOval(position, position, size, size);
    else
      gc.fillRect(position, position, size, size);
  }

  public static void main(String[] args) {
    Application.launch(args);
  }
}
```

Exercice 3. Tri de listes (exercice supplémentaire)

On considère l'interface suivante constituée de trois listes contenant chacune différents items. L'objectif est de trier ces listes en déplaçant les items d'une liste à l'autre (voir Figure 3). Pour cela, on sélectionne l'item que l'on souhaite déplacer d'une liste, puis on clique sur le bouton "déplacer ici" de la liste correspondante.

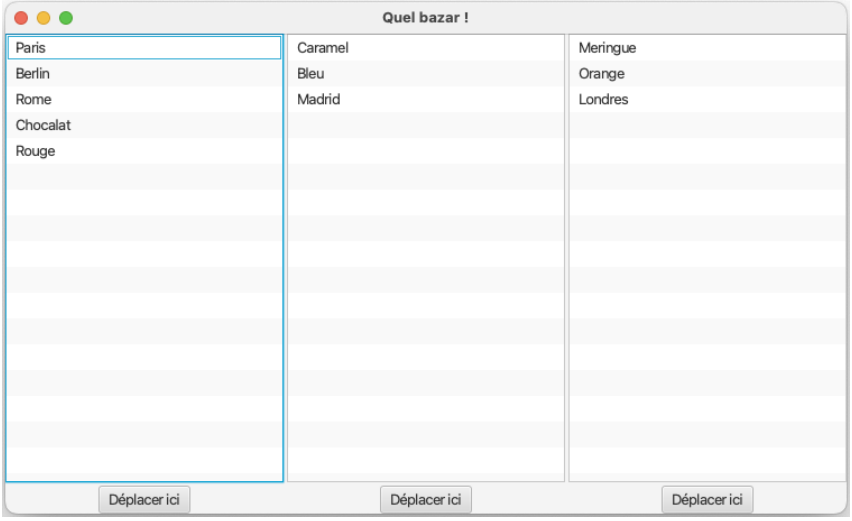

FIGURE 3 – Capture d'écran de l'interface de tri de listes.

```java
public class SortLists extends Application {
        ListView<String> lv1, lv2, lv3;
        Button bt1, bt2, bt3;
        ListView<String> lastSelected;

        public void start(Stage stage) {
                HBox hbox = new HBox(3);

                VBox vb1 = new VBox(3);
                vb1.setAlignment(Pos.CENTER);
                lv1 = new ListView<String>();
                lv1.getItems().addAll("Paris", "Berlin", "Rome", "Chocalat", "Rouge");
                bt1 = new Button("Deplacer ici");
                vb1.getChildren().addAll(lv1, bt1);
                hbox.getChildren().add(vb1);

                VBox vb2 = new VBox(3);
                vb2.setAlignment(Pos.CENTER);
                lv2 = new ListView<String>();
                lv2.getItems().addAll("Caramel", "Bleu", "Madrid");
                bt2 = new Button("Deplacer ici");
                vb2.getChildren().addAll(lv2, bt2);
                hbox.getChildren().add(vb2);

                VBox vb3 = new VBox(3);
                vb3.setAlignment(Pos.CENTER);
                lv3 = new ListView<String>();
                lv3.getItems().addAll("Meringue", "Orange", "Londres");
                bt3 = new Button("Deplacer ici");
                vb3.getChildren().addAll(lv3, bt3);
                hbox.getChildren().add(vb3);

                Scene scene = new Scene(hbox);
                stage.setTitle("Quel bazar !");
                stage.setScene(scene);
                stage.show();
        }

        public static void main(String[] args) {
                Application.launch(args);
        }
}
```

Q 1. Quels sont les deux modèles associés à une ListView ?

Q 2. Utilisez une lambda expression ou une classe interne pour mettre à jour `lastSelected` avec la référence de la dernière liste sur laquelle un item a été sélectionné.

Q 3. Créez maintenant une classe interne pour gérer les événements sur les boutons et mettre correctement à jour les listes.