10. ANNULATION / RÉTABLISSEMENT DES COMMANDES

Nous allons mettre en œuvre l'annulation et le rétablissement des commandes fondés sur les actions. Nous utilisons le patron de conception *Command*.

✍ Créez un paquetage **commands** dans **ensisa.lines**

✍ Dans ce dossier, créez une interface publique Command comportant une opération execute

```
public interface Command {
   void execute();
}
```

✍ Créez une interface publique UndoableCommand qui spécialise l'interface Command et comporte les opérations undo et redo. execute réalise l'opération initialement une seule fois, undo annule l'opération et redo la refait.

```
public interface UndoableCommand extends Command {
   void undo();
   default void redo() {
      execute();
   }
}
```

10.1. GESTIONNAIRE DES COMMANDES ANNULABLES

La classe UndoRedoHistory conserve l'historique des commandes. Deux piles sont nécessaires pour réaliser des annulations et rétablissements en séquence.

Lors de la première exécution d'une commande, celle-ci est empilée dans la pile d'annulation.

L'annulation consiste à dépiler une commande de la pile d'annulation, à invoquer la méthode undo puis à empiler la commande dans la pile de rétablissement.

Le rétablissement réalise l'opération inverse.

✍ Ecrivez la classe UndoRedoHistory qui modélise le gestionnaire des commandes annulables. La méthode execute empile la commande à exécuter et l'exécute. Les propriétés canUndo et canRedo indiquent respectivement si les opérations d'annulation et de rétablissement sont possibles.

```
public class UndoRedoHistory {
   private Stack<UndoableCommand> undoStack;
   private Stack<UndoableCommand> redoStack;
   private boolean inUndoRedo;
   private final BooleanProperty canUndo;
   private final BooleanProperty canRedo;

   public UndoRedoHistory()
   {
      undoStack = new Stack<UndoableCommand>();
      redoStack = new Stack<UndoableCommand>();
      canUndo = new SimpleBooleanProperty(false);
      canRedo = new SimpleBooleanProperty(false);
   }

   public void undo()
   {
      inUndoRedo = true;
      var top = undoStack.pop();
      top.undo();
      redoStack.push(top);
      inUndoRedo = false;
      canUndo.set(!undoStack.isEmpty());
      canRedo.set(true);
   }

   public void redo()
   {
      inUndoRedo = true;
      var top = redoStack.pop();
      top.redo();
```

```
            undoStack.push(top);
            inUndoRedo = false;
            canUndo.set(true);
            canRedo.set(!redoStack.isEmpty());
        }

        public void execute(UndoableCommand command)
        {
            if (inUndoRedo)
                throw new RuntimeException(
                    "Invoking execute within an undo/redo action.");
            // On ne peut réaliser une nouvelle
            // opération pendant l'annulation ou le
            // rétablissement d'une autre
            redoStack.clear();
            // La pile de rétablissement est vidée
            // lorsqu'une nouvelle opération est
            // réalisée
            undoStack.push(command);
            canUndo.set(true);
            canRedo.set(false);
            command.execute();
        }

        public BooleanProperty canUndoProperty() {
            return canUndo;
        }

        public BooleanProperty canRedoProperty() {
            return canRedo;
        }

    }
```

✍ Instanciez le gestionnaire de commandes annulables dans MainController :

```
    private final UndoRedoHistory undoRedoHistory;
```

```
public MainController() {
    document = new Document();
    lineControllers = new HashMap<>();
    selectTool = new SelectTool(this);
    drawTool = new DrawTool(this);
    currentTool = new SimpleObjectProperty<>(selectTool);
    selectedLines = FXCollections.observableSet();
    undoRedoHistory = new UndoRedoHistory();
}
```

✍ Ajoutez une méthode pour exécuter une commande :

```
public void execute(UndoableCommand command) {
    undoRedoHistory.execute(command);
}
```

✍ Déclarez des variables annotées pour référencer les articles de menu **Undo** et **Redo** :

```
@FXML
private MenuItem undoMenuItem;
@FXML
private MenuItem redoMenuItem;
```

✍ Ajoutez des traitants d'événements pour ces articles de menu :

```
@FXML
private void undoMenuItemAction() {
    undoRedoHistory.undo();
}

@FXML
private void redoMenuItemAction() {
    undoRedoHistory.redo();
}
```

✍ Créez des liens entre ces articles de menus et des propriétés du gestionnaire des commandes annulables :

```
public void initialize() {
    linesEditor = new LinesEditor(editorPane);
    setClipping();
    initializeToolPalette();
    initializeMenus();
    observeDocument();
    observeSelection();
}

private void initializeMenus() {
    undoMenuItem.disableProperty().
        bind(undoRedoHistory.canUndoProperty().not());
    redoMenuItem.disableProperty().
        bind(undoRedoHistory.canRedoProperty().not());
}
```

✍ Dans SceneBuilder, renommez le menu **Edit** en **Edition** et insérez des articles de menu **Annuler** et **Rétablir** dans le menu **Edition**. Associez des raccourcis clavier, les fx:id et les traitants d'événement.

✍ Lancez l'application. Les articles de menu **Undo** et **Redo** doivent être désactivés.

10.2.RENDRE ANNULABLE LE DESSIN D'UNE LIGNE

La classe DrawCommand représente la commande de dessin annulable. C'est elle qui crée la ligne qui sera insérée dans le document. L'exécution de la commande ajoute la ligne au document. L'annulation fait l'inverse, elle retire la ligne du document.

✍ Ecrivez la classe DrawCommand dans le paquetage **ensisa.lines.commands** :

```
public class DrawCommand implements UndoableCommand {
    public StraightLine straightLine;
    private Document document;

    public DrawCommand(Document document, double x, double y) {
        this.document = document;
        straightLine = new StraightLine();
```

```
            straightLine.setStartX(x);
            straightLine.setStartY(y);
            straightLine.setEndX(x);
            straightLine.setEndY(y);
        }

        @Override
        public void execute() {
            document.getLines().add(straightLine);
        }

        @Override
        public void undo() {
            document.getLines().remove(straightLine);
        }
    }
```

✍ Modifiez l'outil de dessin pour qu'il crée la commande annulable :

```
    public void mousePressed(MouseEvent event) {
        if (event.isPrimaryButtonDown()) {
            var command = new DrawCommand(mainController.getDocument(),
                event.getX(), event.getY());
            currentLine = command.straightLine;
            mainController.execute(command);
            state = State.drawing;
        }
    }
```

✍ Lancez l'application. Les opérations de dessin peuvent être annulées et rétablies en cascade.

10.3.RENDRE ANNULABLE LA SUPPRESSION

✍ Ecrivez la classe DeleteCommand dans le paquetage **ensisa.lines.commands** :

```
    public class DeleteCommand implements UndoableCommand {
```

```
    private MainController mainController;

    public DeleteCommand(MainController mainController) {
      this.mainController = mainController;
    }
  }
```

✍ Lorsque l'utilisateur annule l'opération de suppression des lignes sélectionnées, la commande doit rétablir la liste des lignes qui figuraient dans le document avant la suppression. La classe DeleteCommand doit donc mémoriser la liste des lignes avant suppression :

```
  public class DeleteCommand implements UndoableCommand {
    private MainController mainController;
    private List<StraightLine> savedLines;

    public DeleteCommand(MainController mainController) {
      this.mainController = mainController;
      savedLines = new ArrayList<>(mainController.getDocument().getLines());
    }
  }
```

✍ Lorsque l'utilisateur rétablit l'opération de suppression des lignes sélectionnées, la commande doit connaitre les lignes qui étaient sélectionnées dans le document avant la suppression. La classe DeleteCommand doit donc mémoriser la collection des lignes sélectionnées avant suppression :

```
  public class DeleteCommand implements UndoableCommand {
    private MainController mainController;
    private List<StraightLine> savedLines;
    private Set<StraightLine> savedSelectedLines;

    public DeleteCommand(MainController mainController) {
      this.mainController = mainController;
      savedLines = new ArrayList<>(mainController.getDocument().getLines());
      savedSelectedLines = new HashSet<>(mainController.getSelectedLines());
    }
  }
```

✍ Ecrivez la méthode qui réalise la suppression des lignes sélectionnées. Elle efface la sélection courante avant de supprimer les lignes sélectionnées précédemment du modèle :

```
@Override
public void execute() {
    mainController.deselectAll();
    mainController.getDocument().getLines().removeAll(savedSelectedLines);
}
```

✍ Ecrivez la méthode qui réalise l'annulation des lignes supprimées. Elle efface la sélection courante, la liste courante des lignes du modèle, restaure l'état du modèle avant suppression puis restaure la sélection avant suppression :

```
@Override
public void undo() {
    mainController.deselectAll();
    mainController.getDocument().getLines().clear();
    mainController.getDocument().getLines().addAll(savedLines);
    mainController.getSelectedLines().addAll(savedSelectedLines);
}
```

✍ Dans MainController, déclarez une variable annotée pour référencer l'article de menu **Delete** :

```
@FXML
private MenuItem deleteMenuItem;
```

✍ Ajoutez un traitant d'événements pour cet article de menu :

```
@FXML
private void deleteMenuItemAction() {
    undoRedoHistory.execute(new DeleteCommand(this));
}
```

✍ Créez un lien entre cet article de menus et le contrôleur :

```
    private void initializeMenus() {
        undoMenuItem.disableProperty().
            bind(undoRedoHistory.canUndoProperty().not());
        redoMenuItem.disableProperty().
            bind(undoRedoHistory.canRedoProperty().not());
        deleteMenuItem.disableProperty().bind(Bindings.createBooleanBinding(() ->
            selectedLines.isEmpty(), selectedLines));
    }
```

✍ Dans SceneBuilder, modifiez l'article de menu **Delete** dans le menu **Edit**. Associez le traitant d'événement.

✍ Lancez l'application. Les opérations de dessin et les suppressions de lignes peuvent être annulées et rétablies en cascade.

11.INSPECTEUR

✍ Dans SceneBuilder, ajouter un composant GridPane dans la partie droite de BorderPane

✍ Dans la cellule (ligne 0, colonne 0), mettez un Label "Epaisseur de ligne"

✍ Dans la cellule (ligne 0, colonne 1), mettez un TextField

✍ Dans la cellule (ligne 1, colonne 0), mettez un Label "Couleur"

✍ Dans la cellule (ligne 1, colonne 1), mettez un ColorPicker

✍ Sélectionnez la ligne 0 et mettez **USE_COMPUTED_SIZE** dans **Pref Height**

✍ Sélectionnez la ligne 1 et mettez **USE_COMPUTED_SIZE** dans **Pref Height**

✍ Sélectionnez la ligne 2 et mettez **ALWAYS** dans **Vgrow**

✍ Sélectionnez la colonne 0 et mettez **USE_COMPUTED_SIZE** dans **Pref Width**

✍ Sélectionnez **GridPane** et mettez **Margin** à (10, 10, 10, 10), **Hgap** et **Vgap** à 5

✍ Dans la classe MainController, déclarez des variables annotées :

```
@FXML
private TextField lineWidthTextField;
@FXML
private ColorPicker colorPicker;
```

✍ Dans la classe MainController, déclarez des traitants d'événement :

```
@FXML
```

```
    private void lineWidthTextFieldAction() {
    }

    @FXML
    private void colorPickerAction() {

    }
```

✍ Dans SceneBuilder, associez ces variables et traitants d'évènement aux composants

11.1.EPAISSEUR DE LIGNE

✍ Dans la classe MainController, observez les changements apportés à la sélection pour déterminer l'épaisseur de ligne commune :

```
    public void initialize() {
        linesEditor = new LinesEditor(editorPane);
        setClipping();
        initializeToolPalette();
        initializeMenus();
        initializeInspector();
        observeDocument();
        observeSelection();
    }

    private void initializeInspector() {
        selectedLines.addListener(new SetChangeListener<StraightLine>() {
            @Override
            public void onChanged(Change<? extends StraightLine> change) {
                lineWidthTextField.setText(findCommonStrokeWidth());
            }

            private String findCommonStrokeWidth() {
                boolean foundOne = false;
                double width = 0.0;
                for (var l : selectedLines) {
                    if (!foundOne) {
                        width = l.getStrokeWidth();
```

```
                foundOne = true;
            } else {
                if (width != l.getStrokeWidth())
                    return "";
            }
        }
        return foundOne ? String.valueOf(width) : "";
    }
});
}
```

✍ Traiter la validation du champ de saisie par l'utilisateur :

```
@FXML
private void lineWidthTextFieldAction() {
    try {
        var value = Double.parseDouble(lineWidthTextField.getText());
        if (value >= 1.0) {
            selectedLines.forEach(straightLine -> {
                straightLine.setStrokeWidth(value);
            });
        }
    } catch (NumberFormatException ex) {
    }
}
```

✍ Lancez l'application

11.2.COULEUR

✍ Dans la classe MainController, observez les changements apportés à la sélection pour déterminer la couleur commune :

```
private void initializeInspector() {
    selectedLines.addListener(new SetChangeListener<StraightLine>() {
        @Override
        public void onChanged(Change<? extends StraightLine> change) {
```

```
            lineWidthTextField.setText(findCommonStrokeWidth());
            colorPicker.setValue(findCommonColor());
        }

    private Color findCommonColor() {
        boolean foundOne = false;
        Color color = Color.TRANSPARENT;
        for (var l : selectedLines) {
            if (!foundOne) {
                color = l.getColor();
                foundOne = true;
            } else {
                if (!color.equals(l.getColor()))
                    return Color.TRANSPARENT;
            }
        }
        return foundOne ? color : Color.TRANSPARENT;
    }
```

✍ Traiter la sélection d'une couleur par l'utilisateur :

```
    @FXML
    private void colorPickerAction() {
        var color = colorPicker.getValue();
        selectedLines.forEach(straightLine -> {
            straightLine.setColor(color);
        });
    }
```

✍ Lancez l'application