ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

ILO
TP Figure Editor

Vous pourrez trouver la plupart des modifications mentionnées ci-dessous dans l'archive /pub/ILO/TPEditor/ComplementsFigureEditor.zip

Compléments et conseils pour l'éditeur de formes géométriques en JavaFX

- `figures`
  - `Figure` : Un changement important est nécessaire dans la classe `Figure` et ses descendantes. Lorsque l'on crée un `Memento<Figure>` de l'état interne du modèle de dessin (`Drawing`) on a besoin de copier toutes les figures dans le Memento. Pour ce faire on utilise la méthode `clone()` de chaque figure qui elle même fait appel au constructeur de copie de chaque figure qui lui même fera appel au constructeur de copie de la classe Figure. Lors de cette copie, il est important que les noeuds JavaFX (`Group`, `Shape`, `Rectangle`) résultant de la copie dans la nouvelle figure à créer soient des instances **distinctes** des noeuds JavaFX à copier. Dans une figure copiée l'attribut `Shape shape` va être ajouté au `Group root` et ce faisant va changer le parent de `shape`. Si `shape` n'est pas copié de manière profonde (avec une instance distincte) cela va changer le graphe de scène dans lequel cette `Shape` est déjà affichée (le `drawingPane`). Le constructeur de copie de la classe `Figure` doit donc être modifié de la manière suivante :

```
protected Figure(Figure figure)
{
    /*
     * Note: the provided figure is supposed to be in a consistent state:
     * Meaning it can not have both edge and fill colors as nulls.
     */
    this(figure.getFillColor(),
         figure.getEdgeColor(),
         figure.lineType,
         figure.lineWidth,
         figure.logger);
    instanceNumber = figure.instanceNumber;
    if (figure.shape == null)
    {
        String message = "null provided shape";
        logger.severe(message);
        throw new NullPointerException(message);
    }
    root.getChildren().add(figure.shape);// A supprimer
    setSelected(figure.selected);// A supprimer


    /*
     * If the copied figure has gone through translation, rotation
     * and rotation these need to be transferred to this new group
     */
    root.setTranslateX(figure.root.getTranslateX());
    root.setTranslateY(figure.root.getTranslateY());
    root.setRotate(figure.root.getRotate());
    root.setScaleX(figure.root.getScaleX());
    root.setScaleY(figure.root.getScaleY());
```

```
        /*
         * CAUTION : figure.shape can NOT be directly transferred to this.shape
         * such as shape = figure.shape;
         * As this.shape will be added to this #root's children which will change
         * the transferred shape's parent and mess up the current JavaFX scene
         * graph.
         * So this.shape need to be a DISTINCT shape from figure.shape with the
         * same characteristics but it can't be done in Abstract class Figure.
         * This needs to be performed in every sub-classes copy constructors
         * Then the newly created shape needs to be
         *  - added to root's children
         *  - #setSelected with figure.selected
         */
}
```

Tous les constructeurs de copie des classes filles de la classe `Figure` devront être modifiés en conséquence. A titre d'exemple voici le constructeur de copie de la classe `Circle` (après modifications) :

```
public Circle(Figure figure) throws IllegalArgumentException
{
    super(figure);
    if (!(figure instanceof Circle))
    {
        String message = "provided figure is not a Circle: "
            + figure.getClass().getSimpleName();
        logger.severe(message);
        throw new IllegalArgumentException(message);
    }
    javafx.scene.shape.Circle figureCircle =
        (javafx.scene.shape.Circle) figure.shape;
    shape = new javafx.scene.shape.Circle(figureCircle.getCenterX(),
                                          figureCircle.getCenterY(),
                                          figureCircle.getRadius());
    root.getChildren().add(shape);
    applyParameters(shape);
    setSelected(figure.selected);
}
```

- Pour créer de `NGon` ou des `Star` je vous conseille d'utiliser des `javafx.scene.shape.Path` pour représenter ces figures. Un `Path` est constitué d'une liste de `PathElement` qui peuvent être, par exemple:
  - `MoveTo(double x, double y)` : pour commencer le dessin en (x,y)
  - `LineTo(double x, double y)` : pour tracer une ligne vers (x, y) à partir du point précédent.
  - `ClosePath()` pour fermer le polygone lorsque l'on est arrivé au dernier point.
  - Vous pourrez donc construire le Path d'un polygone régulier de la manière suivante :

```
center = new Point2D(x, y);
nbSides = sides;
this.radius = radius;
List<PathElement> elements = new ArrayList<>(nbSides + 1);
Path newPath = new Path();
/*
 * First move to center.y - radius to start the path
 */
Point2D point  = new Point2D(center.getX(),
                             center.getY() - radius);
elements.add(new MoveTo(point.getX(), point.getY()));
```

```
        /*
         * Adds LineTo(s) for each side
         */
        // Rotation transform with center as pivot point
        Rotate rotation = new Rotate(360.0 / nbSides,
                                     center.getX(),
                                     center.getY());
        for (int i = 0; i < (sides - 1); i++)
        {
            point = rotation.transform(point);
            elements.add(new LineTo(point.getX(), point.getY()));
        }
        /*
         * Close the path
         */
        elements.add(new ClosePath());
        /*
         * Add all these path elements to the new path
         */
        newPath.getElements().addAll(elements);
        /*
         * Assign to shape
         */
        shape = newPath;
```

Vous pourrez noter que l'on a utilisé ici les propriétés d'une transformation `Rotate` pour faire tourner un point de $\frac{2\pi}{nbSides}$ définissant ainsi les sommets du polygone régulier.

- `enums`
  - `LineType` : Vous pouvez retirer la méthode `fromStroke(BasicStroke stroke)` qui est un reliquat de la version précédente de cet éditeur et n'est plus utilisée.

- `application`
  - `panels`
    - Contient une nouvelle `SizeDialogPane` qui pourra être utilisée dans `Controller#onSetHistorySizeAction` pour changer la taille de l'`HistoryManager`. A ce propos, il faudra sans doute ajouter une méthode `int size()` à la classe `HistoryManager` du package `history`.
  - `Controller`
    - Il serait judicieux d'ajouter un callback `onDemoAction(ActionEvent event)` associé à un `MenuItem` dans le menu *Figures* par exemple permettant d'ajouter les figures que vous avez créées sans passer par les `xxxCreationTool` du package `tools.creation`. Ceci vous permettra de montrer / tester rapidement les figures que vous aurez créées :

```
    @FXML
    public void onDemoAction(ActionEvent event)
    {
      logger.info("Demo action triggered");
      historyManager.record();
      NGon ngon1 = new NGon(drawingModel.getFillColor(),
                            drawingModel.getEdgeColor(),
                            drawingModel.getLineType(),
                            drawingModel.getLineWidth(),
                            logger,
                            100.0,
                            100.0,
                            50.0,
                            6);
```

```
            // ...
            drawingModel.add(ngon1);
            // ...
        }
```

- `history`
  - `HistoryManager`: Ajouter une méthode `public int size()` car vous en aurez besoin dans l'action `onSetHistorySizeAction(ActionEvent event)` du `Controller`.
- `tools`
  - `TransformTool` : Vous aurez besoin d'ajouter un attribut `HistoryManager<Figure> historyManager` et de l'initialiser dans le constructeur de cet outil pour pouvoir ensuite créer un `Memento<Figure>` **avant** chaque nouvelle transformation avec `historyManager.record();` afin que les transformations appliquées puissent être annulées ou refaites avec les actions `onUndoAction(ActionEvent event)` et `onRedoAction(ActionEvent event)` du `Controller`.
  - Pour créer des `NGon` et des `Star` vous pourrez utiliser les `ScrollEvent` (les évènements provenant de la roulette de la souris) pour spécifier le nombre de côtés des `NGon` ou le nombre de branches des `Star`. Pour ce faire, vous pourrez simplement ajouter une méthode `void handleScrollEvent(ScrollEvent event)` à votre `[NGon|Star]CreationTool`:

```
public void handleScrollEvent(ScrollEvent event)
{
    scrollAmount += event.getDeltaY();
    int deltaSide = (int) Math.round(scrollAmount / scrollScale);
    // ...
                sidedPolygon.setNbSides(nbSides);
                scrollAmount = 0.0;
    // ...
    /*
     * ScrollEvent NEEDS to be consumed in order NOT to bubble up
     * in the drawingPane where it might trigger a content scroll
     */
    event.consume();
}
```

Vous pourrez alors enregistrer cette méthode en tant que `EventHandler<ScrollEvent>` sur la racine de la figure (`Group root`, puisque l'attribut `shape` risque d'être changé par cette opération) avec `figure.getRoot().setOnScroll(this::handleScrollEvent);` où l'on utilise une référence de méthode (ici `this::handleScrollEvent`). Et vous pourrez dé-enregistrer cette méthode une fois l'opération terminée avec `figure.getRoot().removeEventHandler(ScrollEvent.ANY, this::handleScrollEvent);`

Les étapes à Compléter en premier

Voici la liste des étapes à compléter en premier pour obtenir un éditeur en état de créer des `Circle` dans la zone de dessin.

Type	#	Description	Location
TODO	001	Figure#hasFillColor ...	/figures/Figure.java
TODO	002	Figure#getFillColor	/figures/Figure.java
TODO	003	Figure#setFillColor ...	/figures/Figure.java
TODO	004	Figure#hasEdgeColor ...	/figures/Figure.java
TODO	005	Figure#getEdgeColor ...	/figures/Figure.java
TODO	006	Figure#setEdgeColor ...	/figures/Figure.java
TODO	007	Figure#setLineType ...	/figures/Figure.java
TODO	008	Figure#setLineWidth ...	/figures/Figure.java
TODO	009	Figure#setSelected ...	/figures/Figure.java
TODO	010	Figure#applyParameters ...	/figures/Figure.java
TODO	011	Figure#updateSelectionFrame ...	/figures/Figure.java
TODO	012	Figure#equals(Object) ...	/figures/Figure.java
TODO	013	Controller#initialize: setting up #drawingModel	/application/Controller.java
TODO	014	Controller#initialize: Binds properties of UI elements to #drawingModel …	/application/Controller.java
TODO	015	Controller#initialize: Setting up #historyManager …	/application/Controller.java
TODO	016	Setup #shapeTypeComboBox …	/application/Controller.java
TODO	017	Setup #lineTypeCombobox …	/application/Controller.java
TODO	018	Controller#initialize: Setup #useFillColor, #useEdgeColor, #fillColorPicker and #edgeColorPicker	/application/Controller.java
TODO	019	Controller#initialize: Setup #lineWidthSpinner with a new SpinnerValueFactory	/application/Controller.java
TODO	020	Controller#initialize: Setup #figuresListView with	/application/Controller.java
TODO	021	Controller#initialize: Setup #messagesLabel with empty or null message	/application/Controller.java
TODO	022	Controller#setTools ...	/application/Controller.java
TODO	023	AbstractCreationTool#createFigure	/tools/creation/AbstractCreationTool.java
TODO	024	AbstractCreationTool#updateFigure	/tools/creation/AbstractCreationTool.java
TODO	025	AbstractCreationTool#cancelFigure	/tools/creation/AbstractCreationTool.java
TODO	026	AbstractCreationTool#terminateFigure ...	/tools/creation/AbstractCreationTool.java
TODO	027	Drawing#initiateFigure ...	/figures/Drawing.java