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

ILO
TP Figure Editor

Introduction

Le but de ce TP qui s'étalera sur l'ensemble des séances restantes est de vous faire travailler en binôme sur une véritable application : Un mini éditeur de formes géométriques.
L'éditeur est composé :

- D'un modèle de dessin gérant le dessin de plusieurs formes géométriques (Cercle, Ellipse, Rectangle, Rectangle arrondi, Polygone, Polygone régulier et Etoile).
- D'une interface graphique permettant
  - De commander les paramètres du modèle de dessin (type de forme, couleurs de trait et de remplissage, type de trait (continu, pointillé, sans trait) et épaisseur du trait.
  - De dessiner les formes gérées par le modèle de dessin.
  - D'effacer l'ensemble des figures
  - De sélectionner des figures puis d'éditer les figures sélectionnées :
    - Déplacement, rotation, facteur d'échelle.
    - Changement de style.
    - Réordonnancement des figures.
    - Destruction des figures sélectionnées.
  - D'afficher dans un panneau d'informations les informations relatives à la figure située sous le curseur de la souris dans la zone de dessin.
  - De filtrer l'affichage des figures suivant plusieurs critères (type de figure, couleur de remplissage ou de trait, types de trait).
  - De gérer les undos / redos.

Vous pourrez trouver une ébauche du code à réaliser dans l'archive :
/pub/ILO/TP-Editor/FiguresEditor.zip
Documentation Java : http://docs.oracle.com/javase/8/docs/api/

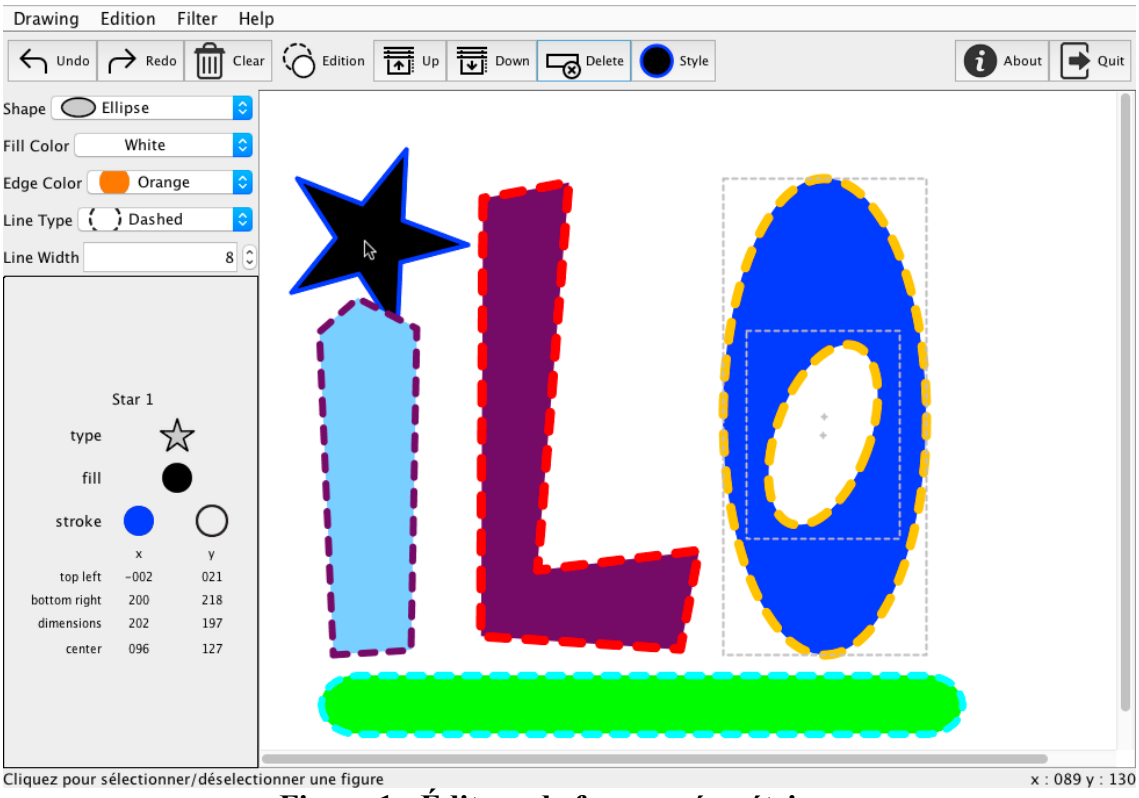

Figure 1 : Éditeur de formes géométriques

Figures

La classe Figure qui vous est fournie sera la classe mère de toutes les figures que vous aurez à construire. Les figures sont composées (entre autres) de :

- Shape shape : une forme géométrique à dessiner (au sens de Java)
- Paint edge/fill : les couleurs de remplissage et de trait des figures à dessiner, une figure peut ne pas avoir de remplissage si « fill » est null
- BasicStroke stroke : le style du trait (continu, pointillé), une figure peut ne pas voir de trait si stroke est null.
- Un nom (le type de la figure) et un numéro d'instance unique pour chaque figure du même type qui permettra de distinguer les figures d'un même type entre elles et d'afficher leur nom dans un panneau d'information dans l'interface graphique.
- 3 transformations affines « translation », « rotation » et « scale » qui peuvent être utilisées pour transformer une figure. Elles sont donc combinées avant le dessin effectif de la figure dans son contexte graphique pour effectivement transformer la figure.
- Un état de sélection indiquant si la figure est sélectionnée ou non.

Les figures répondent aux commandes suivantes :

- Création d'une nouvelle figure.
- Déplacement du dernier point de la figure (ce qui permet de créer des figures de tailles non nulles à la souris). Cette méthode pourra être assortie de toutes les méthodes nécessaires à la création d'une nouvelle figure à l'aide d'évènements souris.
- Dessin de la figure dans un contexte graphique (Graphics2D) en utilisant les shape, edge, fill et stroke ce qui permettra au modèle de dessin de dessiner l'ensemble des figures dans un JPanel de dessin. On rajoutera le dessin de la boite englobante de la figure si celle-ci est sélectionnée.
- Point contenu : détermine si un point donné est contenu à l'intérieur de la figure. Ceci permettra plus tard de remplir un panneau d'informations relatives à une figure située sous le pointeur de la souris dans la zone de dessin ou de sélectionner / désélectionner une figure.
- Transformation de la figure : translation, rotation, facteur d'échelle.
- Plus des accesseurs pour les différents attributs qui peuvent être utilisés et/ou modifiés depuis l'interface graphique ou le modèle de dessin.

Modèle de dessin

Le modèle de dessin (la classe Drawing) contient une collection de figures, ainsi que des méthodes pour :

- Mettre en place les styles de la prochaine figure à créer (le type de figure, les couleurs de trait et de remplissage et le style du trait). Les couleurs et les types de trait seront fournis par des FlyweightFactories (dans le package utils) qui permettent de ne fournir qu'une seule référence d'une couleur ou d'un style de trait donné afin qu'ils puissent être partagés par plusieurs figures.
- Initier une nouvelle figure à la position d'un point p (qui sera fourni par le pointeur de la souris) et l'ajouter à la collection des figures.
- Récupérer la dernière figure de la collection (celle en cours de construction). Notamment afin de la fournir à un XXXCreationListener qui saura interpréter les actions souris nécessaire à la création d'une nouvelle figure.
- Récupérer la dernière figure de la collection qui contient un point p (pour afficher les caractéristiques de cette figure dans un panneau d'informations, ou pour sélectionner / désélectionner une figure en mode édition).
- Supprimer l'ensemble des figures.
- Fournir un flux de figures afin qu'elles puissent être parcourues (en vue de les dessiner par exemple), mais aussi filtrées par des prédicats afin de n'afficher que les figures correspondant aux prédicats.
    - Mettre en place différents filtres sur ce flux de figures (par type de figure, par couleur de remplissage ou de trait et par type de traits).
- Sélectionner ou désélectionner des figures (en mettant en place un mode édition / par opposition au mode création dans lequel on crée des figures)

- Réaliser diverses opérations sur les figures sélectionnée comme
    - Supprimer les figures sélectionnées
    - Remonter ou descendre les figures sélectionnées dans la liste des figures
    - Etc.…

Le modèle de dessin communique suivant deux modes :

- Le modèle de dessin est piloté par les différents contrôleurs (xxxListener) associés aux diverses actions de l'interface graphique.
- Le modèle de dessin est un Observable qui met à jour son ou ses Observers lorsque son état interne change (lorsqu'une figure est ajoutée, modifiée, supprimée ou lorsque les filtres changent). Le JPanel de dessin (DrawingPanel) est donc un Observer du modèle de dessin.

Infrastructure de communication

Certains widgets comme les boutons (JButton), les listes déroulantes (JComboBox) ou les items de menus (JMenuItem/JCheckBoxMenuItem) peuvent modifier le modèle de dessin si on leur attache un contrôleur adéquat : un *ActionListener* pour un bouton ou un item de menu et un *ItemListener* pour une liste déroulante. Ces widgets produisent des événements de haut niveau (*ActionEvent* pour un bouton ou un item de menu, *ItemEvent* pour une liste déroulante). Lorsque l'on attache un contrôleur à un widget, celui-ci peut être unique (à savoir uniquement attaché à ce widget), auquel cas il pourra être implémenté en tant que classe anonyme (ou lambda-expression) directement dans l'ajout du contrôleur (dans l'appel de la méthode addXXXListener). Mais si l'on souhaite réutiliser un même contrôleur sur plusieurs widgets ou à plusieurs endroits de l'application on les implémentera en tant que classes internes (dans la fenêtre principale par exemple).
Une extension de « ActionEvent » est fournie par l'interface « Action » qui combine une description de l'action associée à une icône et la gestion des ActionEvents dans une seule classe afin que cette action puisse être attachée à plusieurs widgets comme un bouton ET un item de menu (voir Figure 4, page 5). Plusieurs « Actions » (certaines à compléter) vous sont fournies et pourront être associées à vos boutons et items de menu comme par exemple :

- QuitAction : action à réaliser pour quitter l'application
- UndoAction : action à réaliser pour annuler le dessin de la dernière figure
- RedoAction : action à réaliser pour annuler la dernière annulation
- ClearAction : action à réaliser pour effacer l'ensemble des figures
- AboutAction : action à réaliser pour afficher une boite de dialogue « A propos … » de l'application.
- ShapeFilterAction : pour mettre en place le filtrage sur les différents types de figures.
- LineFilterAction : pour mettre en place le filtrage des figures sur la base des différents types de trait.
- …

De la même manière un certain nombre de listeners qui pourront servir de contrôleurs aux listes déroulantes (de JLabeledComboBox[1] dans notre cas) sont définis en tant classes internes à la fin de la classe widgets.EditorFrame :

- ShapeItemListener : Pour écouter les évènements de la liste déroulante permettant de choisir le type de figure à créer
- ColorItemListener : Pour écouter les évènement de la liste déroulante permettant de choisir les couleurs de remplissage ou de trait des figures
- EdgeTypeListener : Pour écouter les évènements de la liste déroulante permettant de choisir le type de trait des figures (plein, pointillé, ou aucun).
- EdgeWidthListener : Pour écouter les évènements d'un éventuel spinner ou slider permettant de choisir l'épaisseur du trait.

D'autres widgets comme les JPanel ou les JLabel permettent d'afficher des informations (respectivement du dessin ou du texte). La zone de dessin dans notre cas un JPanel : DrawingPanel.

[1] Classe héritière de JComboBox permettant d'utiliser une icône dans les items de la liste.

Par ailleurs tous les widgets émettent des événements de bas niveau dont font partie les événement souris (MouseEvent) que nous souhaitons utiliser pour dessiner nos figures. La construction graphique des figures dans le panel de dessin fera donc appel à un ou plusieurs MouseListener / MouseMotionListener. Néanmoins, chaque type de figure peut nécessiter un contrôle des évènements souris différent. La classe « figures.listeners.creation.AbstractCreationListener » fournit un squelette de base des listeners que vous aurez à créer pour construire graphiquement les différents types de figures.

Plus généralement, le package « figures.listeners » contient les différents Mouse[Motion]Listeners utilisés pour gérer les évènement souris afin de créer ou modifier des figures.

Ces différents éléments permettent de mettre en place une architecture Model / View / Controller vue en cours, pour réaliser la construction, le dessin et la modification des figures :

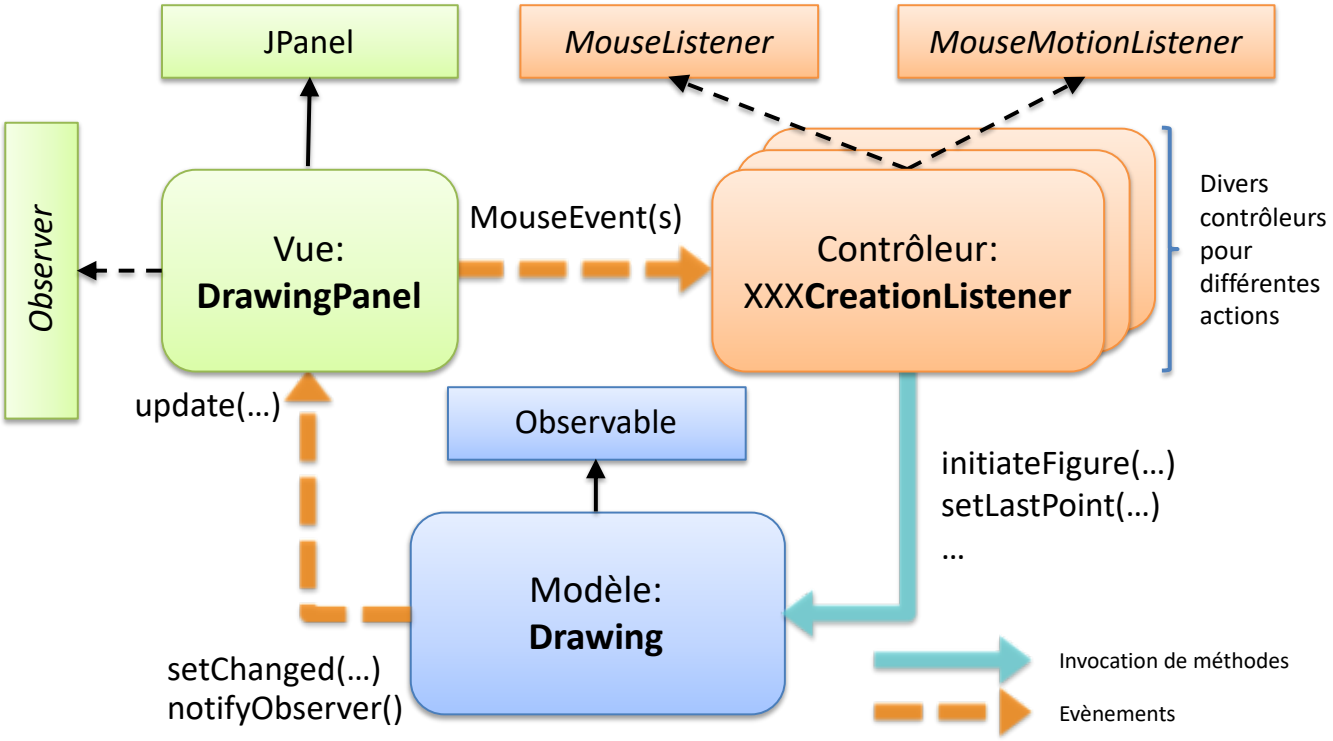

Figure 2 : le MVC entre les modèle de dessin et l'interface graphique

Architecture de l'interface graphique

L'interface graphique à construire dans la classe EditorFrame se structure de la manière suivante :

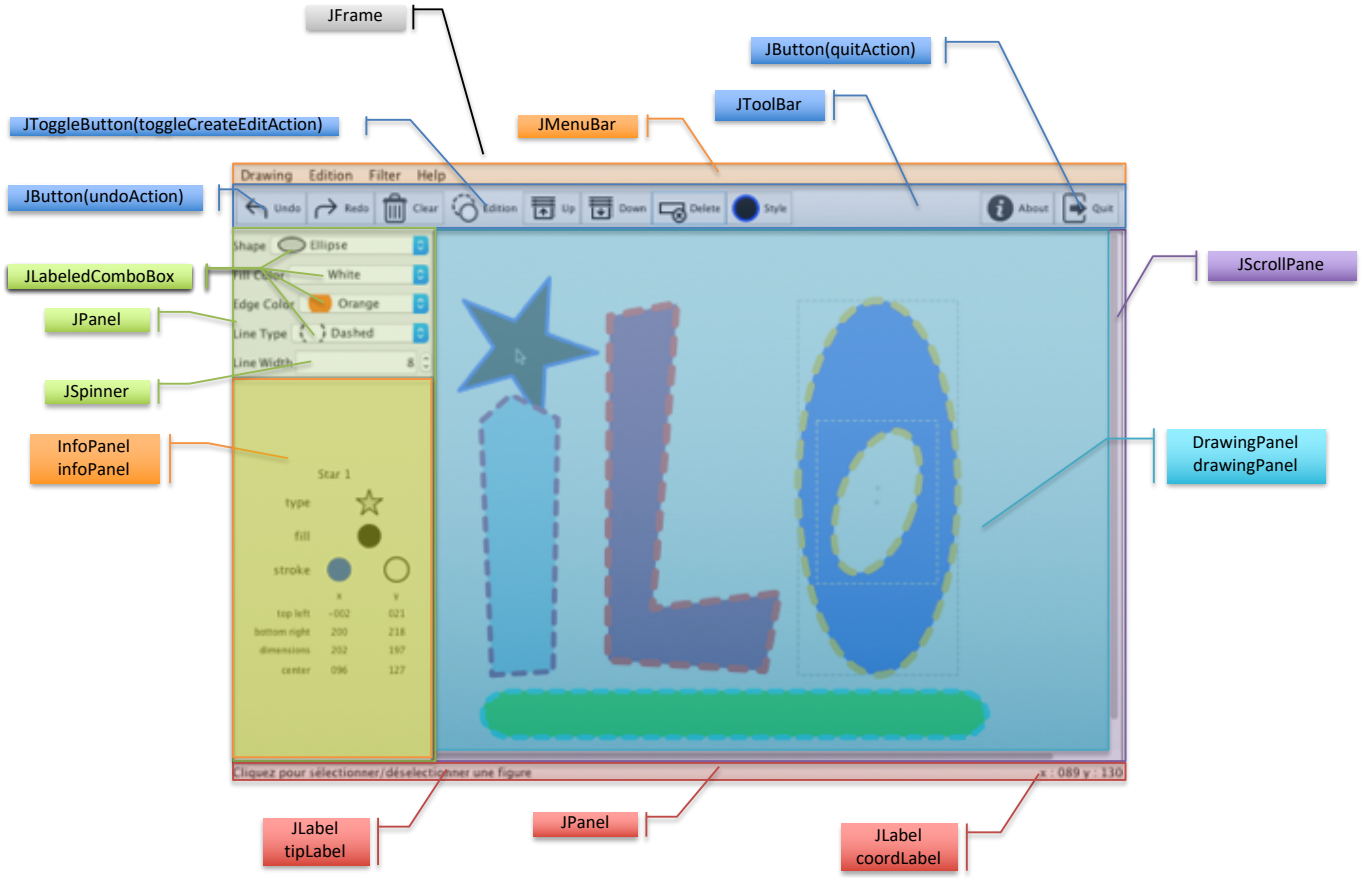

Figure 3 : Structure de l'interface graphique

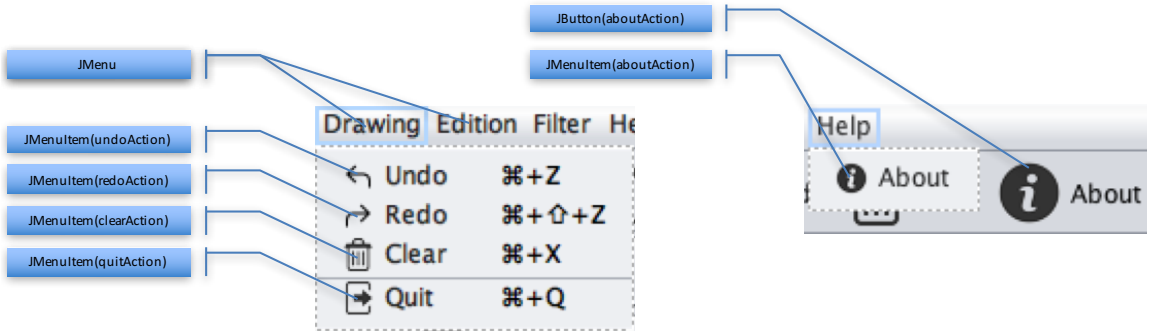

Figure 4 : Actions dans les menus et boutons

La fenêtre principale (EditorFrame héritière de JFrame)

La fenêtre principale contient les différents widgets qui composent l'interface graphique. Ces widgets sont organisés (placés et dimensionnés) en utilisant un LayoutManager, en l'occurrence un BorderLayout permettant de placer les widgets en utilisant les directions cardinales (North : en haut, South : en bas, East : à droite, West : à gauche et Center : au centre).

Barre de menus (JMenuBar)

La barre de menu contient trois menus et chaque item de menu est associé à un « ActionListener » anonyme ou bien à une « Action » (par exemple « Undo » à une « UndoAction ») :

- Le menu Drawing doit contenir les items (de type JMenuItem) suivants :
  - MagicDraw : pour réaliser une démonstration rapide des différentes figures implémentées.
  - Undo : pour annuler la dernière opération sur les figures.
  - Redo : pour refaire la dernière opération annulée sur les figures.
  - Clear : pour effacer l'ensemble des figures.
  - Quit : pour quitter l'application.
- Le menu Edit contient les items (de type JMenuItem ou JCheckBoxMenuItem) suivants :

    - Edition : Un JCheckBoxMenuItem permettant de passer en mode « édition des figures » ou de revenir au mode « création des figures ».
    - Up : Pour remonter les figures sélectionnées en haut de la liste des figures.
    - Down : Pour descendre les figures sélectionnées en bas de la liste des figures.
    - Delete : Pour supprimer les figures sélectionnées
    - Style : Pour appliquer aux figures sélectionnées les styles courants de remplissage et de trait.
- Le menu Filter contient les items (de type JCheckBoxMenuItem) suivants :
    - Filtering : pour mettre en place ou annuler le filtrage des figures
    - Sous menu Figures :
        - Contient un item pour mettre en place un filtre pour chaque type de figure.
    - Sous menu Colors :
        - Fill Color : pour mettre en place le filtrage des figures sur la base de leur couleur de remplissage (couleur à choisir dans une boite de dialogue de choix de couleurs).
        - Edge Color : pour mettre en place le filtrage des figures sur la base de leur couleur de trait (couleur à choisir dans une boite de dialogue de choix de couleurs).
    - Sous menu Strokes :
        - Contient un item pour mettre en place un filtre pour chaque type de trait de figure.
- Help
    - About : pour afficher la boite de dialogue « A propos … »

Barre d'outils (JToolBar)

La barre d'outils contient des boutons associés aux mêmes actions que dans les menus :

- Undo
- Redo
- Clear
- Edition (toggle)
- Up
- Down
- Delete
- Style
- About
- Quit (à droite de la barre d'outils)

Panneau d'outils (JPanel à gauche)

Le panneau d'outils contient les différentes listes déroulantes (en l'occurrence des JLabeledComboBox qui vous sont fournis) permettant de choisir les options de dessin et d'afficher des informations sur les figures :

- Choix du type de figure.
- Choix de la couleur de remplissage.
- Choix de la couleur de trait.
- Choix du type de trait.
- Épaisseur du trait (JSpinner).
- Un panneau d'information (InfoPanel). Le panneau d'information contient des labels mis à jour lorsque le pointeur de la souris se trouve au-dessus d'une figure dans le panel de dessin.

Les JLabeledComboBox sont des combobox associés à un titre et dont chacun des éléments est composé d'un texte associé à une image. L'image chargée pour un item de la liste correspond à un fichier image (.png) portant le même nom que le titre de l'item et se trouvant dans le package « images ».

Barre d'état (JPanel en bas)

La barre d'état contient deux JLabel :

- tipLabel dans lequel le xxxCreationListener de la figure en cours de création affiche ses conseils utilisateur.
- coordLabel dans lequel le panel de dessin affiche les coordonnées courantes du pointeur de la souris.

Zone de dessin (DrawingPanel à droite)

Le DrawingPanel est un observateur du modèle de dessin (Drawing) mais aussi un MouseListener et un MouseMotionListener afin de :

- Le mettre à jour : redessiner les figures du modèle de dessin lorsque celui-ci change.
- Afficher les coordonnées du pointeur de la souris dans un label de la barre d'état.
- Lui affecter le XXXCreationListener relatif à la figure en cours de création ou bien en cours de modification. Les différents listeners affectés à la zone de dessin pour écouter les évènements souris sont les suivants :
    - AbstractFigureListener : classe mère de tous les listeners concernant la manipulation de figures
        - SelectionFigureListener : listener simple permettant de changer une figure située sous le curseur de l'état non sélectionné à l'état sélectionné ou inversement en cliquant dans la figure.
        - AbstractCreationListener : classe mère de tous les listeners destinés à la création de nouvelles figures
            - RectangulaShapeCreationListener : listener utilisé lors de la création de toutes les figures inclues dans un rectangle : Ellipse, Rectangle, RoundedRectangle que l'on peut dessiner en pressant le bouton 1 de la souris, puis en tirant la souris (en maintenant le bouton enfoncé), puis en relâchant le bouton à une position différente du point de départ.
            - …
        - AbstractTransformShapeListener : classe mère de tous les listeners destinés à transformer géométriquement les figures sélectionnées (grâce aux transformations affines « translation », « rotation », « scale » contenues dans les figures). Pour pouvoir utiliser plusieurs Transform listeners simultanément, il suffit de leur affecter un masque lié à une touche de contrôle (par exemple Shift-Click pour tourner et Alt-Click pour changer l'échelle).
            - MoveShapeListener : listener utilisé pour déplacer les figures
            - RotateShapelistener : listener utilisé orienter les figures
            - ScaleShapeListener : listener utilisé pour changer l'échelle des figures

Travail à réaliser

Ce travail est à réaliser en binôme. Si vous souhaitez travailler en monôme, nous adapterons le barème en conséquence. Les paragraphes suivants décrivent le contenu des différents packages du projet *en précisant en italique les tâches à effectuer dans chacun d'entre eux*.

Package « figures »

Le package figures contient :

- Le modèle de dessin **Drawing**, *qu'il vous faudra compléter pour le rendre fonctionnel. N'oubliez pas que le modèle de dessin est un Observable observé par la zone de dessin, il ne faudra donc pas oublier de déclencher la méthode update() à chaque fois que le modèle de dessin est modifié par une opération afin que la zone de dessin soit notifiée de ces changements et puisse redessiner.*
- Les figures géométriques (classes filles de la classe abstraite **Figure**). Vous avez déjà la classe Ellipse, *il vous faudra ajouter les autres classes de figures géométrique comme **Circle**, **Rectangle**, **RoundedRectangle**, **Polygon**, **NGon** et **Star**.*
- Un certain nombre d'enums pour représenter des constantes symboliques des figures
    - **FigureType** : qui contient une constante symbolique pour chaque figure et *qu'il vous faudra compléter puisque c'est cette classe qui sera utilisée pour initier une nouvelle figure en fonction du type de figure choisi et initier un creationListener adapté à ce type de figure*.
    - **LineType** : qui contient les constantes symboliques de types de contour des figures (trait plein, pointillé ou aucun trait)
    - **PainToType** : qui contient deux constantes symboliques FILL & EDGE permettant de définir à quelle partie d'une figure doit s'appliquer une couleur (le remplissage ou le trait) et qui permet d'appliquer une couleur soit au remplissage soit au trait du modèle de dessin.
- Un certains nombres de listeners tous héritiers de la classe abstraite AbstractFigureListener qui vous permettront soit de
    - Dessiner de nouvelles figures à la souris avec les classes héritières de AbstractCreationListener dont vous avez déjà un exemple avec la classe RectangularShapeCreationListener que vous pourrez utiliser pour créer des Ellipse ou des Rectangle.
        - *Il faudra donc créer de nouveaux CreationListeners adapté aux types de figures que vous ajouterez.*
    - Modifier les figures du dessin avec les classes héritières de AbstractTransformShapeListener
        - *Il faudra donc créer les TransformListeners nécessaires pour pouvoir déplacer, tourner ou changer l'échelle des figures sélectionnées*.
    - *Il vous faudra aussi créer un Listener permettant sélectionner / désélectionner les figures* situées sous le curseur de la souris afin justement de leur appliquer des opérations comme les transformations mentionnées au point précédent, mais aussi d'autres opérations comme appliquer le style courant (couleurs de remplissage et de trait, type et épaisseur de trait) aux figures sélectionnées, ou encore remonter ou descendre les figures sélectionnées dans la liste des figures du modèle de dessin.

Package « filters »

Le package doit contenir l'ensemble des filtres que vous pourrez appliquer sur un flux de figures tel que celui fourni dans la méthode stream() de la classe figures.Drawing. Ce package contient :

- Une classe abstraite générique **FigureFilter<T>** permettant de tester un prédicat sur un élément de type T (qui pourra être dans notre cas un type de figure (FigureType), une couleur de trait ou de remplissage (Paint) ou bien un type de contour (LineType). *Il vous faudra donc créer des classes filles afin de réaliser les différents filtrages de figures*.
- Une classe générique **FigureFilters<T>** qui représente un filtre composite sur des éléments de type T, c'est-à-dire un filtre composé d'autres filtres et dont le prédicat est considéré comme vrai si <u>au moins un</u> prédicat des filtres qui le compose est vrai. Ce filtre composite pourra être utilisé pour filtrer de multiples types de figures ou de multiples type de contours de figures.

Package « history »

Le package history contient les classes nécessaires à la réalisation d'un système de gestion des undo/redo implémenté sous la forme d'un pattern Memento. Il contient les classes suivantes :

- **Prototype<T>** déclare une interface pour réaliser clonage d'un objet. Les figures implémentent Prototype<Figure>.
- **Originator<E extends Prototype<E>>** déclare l'interface d'un objet pouvant créer et mettre en place des Mementos. Le modèle de dessin implémente l'interface **Originator<Figure>**.
- **Memento<E>** implémente le memento sauvegardant un état constitué d'une liste d'éléments de type E. La classe **figures.Drawing** implémentant l'interface **Originator<Figure>**, ses mementos seront donc constitués de la liste des figures à dessiner.
- **HistoryManager<E extends Prototype<E>>** gère un historique de undos et de redos permettant :
    - D'enregistrer l'état actuel d'un originator avec la méthode **record()**.
    - De restituer à l'originator le dernier état sauvegardé par la méthode record() avec la méthode **undo()** et accessoirement d'enregistrer l'état courant afin qu'il puisse être restitué plus tard si l'on change d'avis avec la méthode redo().
    - De restituer à l'originator l'état sauvegardé par la méthode undo() avec la méthode **redo()** et accessoirement d'enregistrer l'état courant afin qu'il puisse être restitué plus tard si l'on change d'avis avec la méthode undo().
    - *Il vous faudra compléter la classe HistoryManager afin de gérer des piles de undo et de redo*.

Package « utils »

Le package utils contient de nombreuses classes utilitaires :

- **CColor** étend la class java.awt.Color qui elle-même implémente l'interface java.awt.Paint afin de pouvoir comparer les couleurs entre elles avec un comparateur ce que n'autorise pas la classe Color.
- **FlyweightFactory<T>** fournit un stockage unifié d'éléments de type T : Lorsqu'on lui demande un élément avec la méthode get(T element) pour la première fois il l'enregistre dans une HashMap. Si on lui redemande ce même élément plus tard, il fournira celui qu'il a déjà enregistré en comparant l'élément demandé à ceux qu'il a déjà stocké. La classe FlyweightFactory est utilisée dans factories suivantes :
    - **IconFactory** permet d'obtenir une icône (une image png stockée dans le package « images ») d'après son nom avec la méthode getIcon(String name). Par exemple IconFactory.getIcon("Red") fournira une ImageIcon correspondant au fichier ./images/Red.png.
    - **PaintFactory** permet d'obtenir un Paint (implémenté par la classe Color) soit en utilisant directement un Paint ou bien en utilisant un nom de couleur.
    - **StrokeFactory** permet d'obtenir un BasicStroke (utilisé pour déterminer le type et l'épaisseur de trait lors du dessin) à partir d'un LineType et d'une épaisseur de trait.
- **IconItem** représente les items (une image + un texte) utilisés dans les JLabeledComboBox.
- **Signature** représente une interface implémentée par de nombreuses classes de ce projet et qui permet aux classes qui l'implémentent d'obtenir facilement le nom de la classe ainsi que le nom de la méthode en cours d'utilisation pour fournir cette « signature » dans les messages de debug.
- **Vector2D** est une classe utilitaire de vecteur 2D qui vous apporte un certain nombre d'opérations sur les vecteurs comme les produits scalaire et vectoriel, la norme, l'angle entre deux vecteurs ou la transformation d'un vecteur par une transformation affine.

Package « widgets »

Le package widgets regroupe l'ensemble des éléments graphiques de l'interface utilisateur (GUI) et contient les classes suivantes :

- Un enum **OperationMode** contenant les constantes symboliques *CREATION* et TRANSFORMATION utilisées pour passer l'éditeur du mode création de figures au mode modification des figures dans l'action « toggleCreateEditAction ».

- La classe **DrawingPanel** qui représente le JPanel dans lequel seront dessiné les figures. *Il vous faudra compléter cette classe pour mettre à jour la position du curseur de la souris dans la barre d'état et mettre à jour l'InfoPanel en fonction de la figure située sous le curseur de la souris.*
- La classe InfoPanel qui représente le panneau d'information *qu'il faudra compléter pour afficher les caractéristiques de la figure située sous le curseur de la souris :*
    - *Nom de la figure*
    - *Type de figure*
    - *Couleur de remplissage et de contour*
    - *Type de trait pour le contour*
    - *Les coordonnées de la boite englobante (minX, minY, maxX, maxY)*
    - *La largeur et la hauteur de la figure*
    - *Les coordonnées du centre de la figure*
- La classe **JLabeledComboBox** que vous pourrez utiliser pour créer les liste déroulantes pour le type figures, les couleurs de remplissage et de trait, ainsi que le type de trait afin d'afficher une icône associée au nom.
- La classe **EditorFrame** qui représente la fenêtre principale de l'application. Cette classe contient :
    - La création de l'interface graphique (dans le constructeur de la classe) *que vous pourrez personnaliser en utilisant le WindowBuilder Editor en lieu et place du Java Editor*[2].
    - XxxAction : Un ensemble de classes internes représentant les différentes actions que l'on peut associer à des boutons ou des items de menus *et que vous devrez compléter*. Chacune de ces actions est instanciée de manière finale dans la classe EditorFrame.
    - YyyListener : Un autre ensemble de classes internes représentant les listeners à associer aux différents JComboBox (ou JLabeledComboBox dans note cas) afin de sélectionner le type de figure à créer, les couleurs de remplissage et de trait, le type de trait ainsi que l'épaisseur du trait. *Il vous faudra compléter ces différents listeners et aussi les associer aux différents widgets que vous aurez ajouté dans l'interface graphique*.

Conseils

Vous allez travailler en binôme sur ce projet. Il est suffisamment grand pour que chaque membre du binôme puisse travailler sur sa partie sans créer de conflits de code entre vous. Néanmoins si cela ce produit, vous pourrez avantageusement utiliser la fonction de comparaison de votre IDE en comparant vos deux projets, vous pourrez intégrer (merger) les modifications des sources de l'un ou l'autre des projets. Un exemple de comparaison et de merge de projets est disponible dans /pub/ILO/TP-Editor/EclipseMergeDemo.mp4.

[2] Voir /pub/ILO/TP-Editor/WindowBuilderDemo.mp4 pour une démonstration de WindowBuilder.