TME 8 : sélectionner et modifier les éléments graphiques

Dans ce TME, nous reprenons l'application `PinBoard` (et donc le projet GitLab et le projet Eclipse) de la semaine dernière et nous y ajoutons des fonctionnalités pour sélectionner des éléments graphiques d'une planche, les déplacer, les supprimer, ainsi qu'un mécanisme de copier / coller.

8.1 Sélection

Nous ajoutons un nouvel outil permettant de sélectionner un élément graphique en cliquant dessus, ou plusieurs éléments si la touche *shift* est maintenue enfoncée. Les éléments sélectionnés peuvent ensuite être déplacés à la souris en maintenant le bouton de la souris appuyé. Pour faciliter le repérage des éléments sélectionnés, leur rectangle englobant est dessiné en bleu.

8.1.1 Objet sélection (`Selection`)

La liste des éléments graphiques sélectionnés est une information appartenant à `EditorWindow`. Néanmoins, afin de ne pas surcharger cette classe, nous encapsulons la gestion de ces informations dans un *objet sélection*. La classe `Selection` maintient, dans un attribut, une liste d'objets `Clip` sélectionnés, et possède les méthodes suivantes :

- `public void select(Board board, double x, double y)`
  modélise une sélection simple : la méthode vide d'abord la sélection, puis y ajoute le premier élément graphique de la planche qui contient le pixel à la position spécifiée. Si aucun tel élément n'existe, la sélection reste vide. Pour déterminer si un élément graphique (donc d'interface `Clip`) contient un pixel, nous utiliserons sa méthode `isSelected`.
- `public void toogleSelect(Board board, double x, double y)`
  modélise une sélection multiple : la méthode cherche le premier élément de la planche contenant le pixel spécifié ; cet élément est ajouté à la sélection s'il n'était pas déjà dans la sélection, et est retiré s'il était déjà dans la sélection.
- `public void clear()`
  vide la sélection.
- `public List<Clip> getContents()`
  retourne la sélection.
- `public void drawFeedback(GraphicsContext gc)`
  dessine le rectangle englobant de tous les éléments de la sélection.

⇒ **Travail demandé :** Implantez la classe `pobj.pinboard.editor.Selection` et ajoutez un attribut de classe `Selection` à `EditorWindow`. Testez votre implantation à l'aide de la classe de test `pobj.pinboard.editor.test.SelectionTest` fournie.

8.1.2 Outil de sélection et de déplacement (`ToolSelection`)

Afin d'ajouter un outil « sélection » à notre barre d'outils, nous devons créer une classe `ToolSelection`, obéissant à l'interface `Tool` vue la semaine dernière. Cet outil servira d'abord à sélectionner un ou plusieurs éléments, en appelant des méthodes de l'attribut de classe `Selection` de notre classe `EditorWindow`. Le même outil servira ensuite à déplacer les éléments sélectionnés sur la planche. Plus précisément :

- si l'utilisateur clique sans appuyer sur la touche *shift*, alors `select` est utilisé : il s'agit d'une sélection simple ;
- si l'utilisateur clique en appuyant sur la touche *shift*, alors `toogleSelect` est utilisé : il s'agit d'une sélection multiple ;
- dans tous les cas, les actions `drag` ont pour effet de déplacer simultanément tous les éléments sélectionnés pour suivre les mouvements de la souris tant que le bouton de la souris est enfoncé.

Pour savoir si la touche *shift* est appuyée, il suffit de consulter l'objet `MouseEvent` passé en argument à chaque événement souris.

Notez que `ToolSelection` doit pouvoir accéder au nouvel attribut `Selection` de l'objet `EditorWindow` appelant. Nous avions pour cela déjà prévu un *getter* associé dans `EditorInterface`. Celui-ci était implanté dans `EditorWindow` en retournant `null`, mais il peut maintenant être modifié pour retourner l'attribut `Selection` de `EditorWindow`.

⇒ **Travail demandé :** Implantez la classe `pobj.pinboard.editor.tools.ToolSelection` ; testez-là avec la classe `pobj.pinboard.editor.tools.test.ToolSelectionTest` fournie ; ajoutez un bouton « Select » à la barre de boutons ; connectez l'outil au bouton dans `EditorWindow`.

8.2 Copier et coller

Nous ajoutons maintenant des fonctionnalités de copier / coller et de suppression, utilisables grâce à un menu déroulant « Edit ». Les fonctions de copier / coller utilisent la notion de « presse-papiers », qui est un conteneur invisible d'éléments graphiques. Le menu « Edit » contiendra alors trois actions :

- « Copy » consiste à copier les éléments de la sélection dans le presse-papiers ;
- « Paste » consiste à ajouter une copie des objets du presse-papiers dans la planche ;
- « Delete » consiste à supprimer les éléments sélectionnés de la planche.

Notez les différences importantes entre les objets `Selection` et le presse-papiers :

- Chaque fenêtre d'édition a sa propre sélection, mais le presse-papiers est un objet unique, partagé entre toutes les fenêtres d'édition. Il permet donc de copier d'une planche à une autre.
- La sélection référence les éléments d'une planche, donc modifier un objet de la sélection altère le dessin dans la fenêtre correspondante. Par contre, les éléments sont copiés quand ils sont placés dans le presse-papiers ; ainsi, modifier un objet après l'avoir copié dans le presse-papiers n'influence pas la version de l'objet résidant dans le presse-papiers.

8.2.1 presse-papiers (`Clipboard`)

Objectif pédagogique : motif singleton.

Le presse-papiers est une classe `Clipboard` qui contient, dans un attribut privé, la liste des éléments copiés, et qui a pour méthodes :

- `public void copyToClipboard(List<Clip> clips)`
  qui copie les éléments graphiques en argument dans le presse-papiers.
- `public List<Clip> copyFromClipboard()`
  qui retourne une copie des éléments graphiques du presse-papiers.
- `public void clear()`
  qui vide le contenu du presse-papiers.
- `public boolean isEmpty()`
  qui indique si le presse-papiers est vide.

Pour s'assurer qu'il existe une unique instance du presse-papiers dans toute l'application, nous utilisons le *motif singleton*. Cela signifie que la classe `Clipboard` possède :

- un attribut `Clipboard` statique (privé, comme tous les attributs) : notre unique presse-papiers, créé au chargement de la classe par la machine virtuelle Java ;
- un constructeur `Clipboard()` privé ; il n'y a aucun constructeur publique afin d'empêcher toute autre classe de créer par erreur un presse-papiers (il est nécessaire de définir ce constructeur privé, sinon Java fournit un constructeur par défaut qui est publique) ;

– une méthode publique statique `Clipboard getInstance()` retournant l'attribut `Clipboard` statique, permettant aux autres classes d'accéder au presse-papiers. Ainsi, par exemple, pour copier dans le presse-papiers, un client fera : `Clipboard.getInstance().copyToClipboard(...)`. Deux objets appelant tous deux `Clipboard.getInstance()` sont assurés de référencer la même instance de `Clipboard` (puisqu'il existe une unique instance).

⇒ **Travail demandé :** implantez la classe `pobj.pinboard.editor.Clipboard` et testez-là avec `pobj.pinboard.editor.test.ClipboardTest` ; ajoutez le menu déroulant « Edit », avec les options « Copy », « Paste », « Delete » et implantez les actions associées à ces options.

8.2.2 Observateur (`ClipboardListener`)

Objectif pédagogique : implanter le motif observateur.

Les options « Copy », « Paste » et « Delete » du menu « Edit » restent actives même si les actions correspondantes n'ont aucun sens, par exemple si la sélection ou le presse-papiers est vide. Nous souhaitons maintenant griser ces options quand l'action correspondante est impossible, en utilisant la méthode `setDisable` de `MenuItem`.

Nous nous intéressons ici uniquement au cas de l'option « Paste », la plus intéressante. Celle-ci doit être grisée quand le presse-papiers est vide. La difficulté principale est que toute fenêtre d'édition peut mettre à jour le presse-papiers, soit en y copiant des éléments avec `copyToClipboard`, soit en le vidant avec `clear` ; néanmoins, cette action doit mettre à jour l'option « Paste » de *toutes* les fenêtres, puisque chaque fenêtre a son propre menu « Edit », mais toutes référencent le même presse-papiers. Il nous faut donc un mécanisme flexible permettant à l'objet `Clipboard` d'informer toutes les fenêtres d'un changement d'état. Nous utilisons pour cela le *motif observateur* (ou *listener*).

Avec ce motif, le presse-papiers maintient une liste de *cibles*, obéissant à l'interface `ClipboardListener` :

tme/src/tme8/src/pobj/pinboard/editor/ClipboardListener.java

```
package pobj.pinboard.editor;

public interface ClipboardListener {
   public void clipboardChanged();
}
```

Après chaque changement d'état, le presse-papiers informe toutes les cibles de ce changement en appelant leur méthode `clipboardChanged`. À charge à la méthode `clipboardChanged` de la cible d'interroger le presse-papiers pour connaître son nouvel état et effectuer les actions nécessaires (e.g., griser une option de menu).

Le presse-papiers est enrichi de deux méthodes publiques :

– `public void addListener(ClipboardListener listener)`

– `public void removeListener(ClipboardListener listener)`

permettant aux cibles de s'enregistrer pour recevoir les notifications du presse-papiers, et de se désenregistrer pour ne plus les recevoir. Dans notre cas, une fenêtre `EditorWindow` devra donc obéir à l'interface `ClipboardListener`, s'enregistrer dès sa création auprès du presse-papiers, et se désenregistrer lors de sa fermeture.

⇒ **Travail demandé :** Enrichissez la classe `Clipboard` pour implanter le motif observateur et testez-là avec `pobj.pinboard.editor.test.ClipboardListenerTest` ; servez-vous de l'observateur dans la classe `EditorWindow` pour griser l'option « Paste » des fenêtres d'édition quand nécessaire.

8.3 Bonus : palette de couleurs

Objectif pédagogique : scènes et nœuds graphiques en JavaFX.

Le but de cette extension est d'ajouter un peu de couleur à nos dessins. Nous ajoutons une notion de « couleur courante », qui vient compléter l'outil courant associé à une fenêtre. Une palette de couleurs est simplement une barre de boutons : chaque bouton contient un rectangle coloré `javafx.scene.shape.Rectangle`. L'action associée à un bouton est de changer la couleur courante, ce qui affectera les prochains rectangles et ellipses ajoutés à la planche.

⇒ **Travail demandé :** Ajoutez une palette de couleurs aux fenêtres d'édition et modifiez les outils d'ajout de rectangle et d'ellipse pour tenir compte de la couleur courante. Une extension plus avancée consisterait à permettre aux boutons de changer la couleur des objets sélectionnés.

8.4 Rendu du TME (OBLIGATOIRE)

Vous ferez un *push* sur le serveur GitLab, suivi de la création d'un *tag*.

Dans le champ « Release notes » de votre *tag* vous :

1. fournirez une capture d'écran montrant la sélection des éléments graphiques ;
2. répondrez à la question suivante : pourquoi est-il important, à la question 8.2.2, de désenregistrer les fenêtres d'édition lors de leur fermeture.

Vous préciserez également si vous avez développé l'extension proposée en bonus, ou une autre extension de votre invention.

Vous vous assurerez également que l'intégration continue sous GitLab passe avec succès les tests des TME 7 et 8.