TP2 : Manipulation directe

Objectifs :

- Savoir dessiner des formes géométriques en utilisant les classes *JavaFX* de haut niveau permettant des interactions,
- savoir programmer les interactions de manipulation directe suivantes : pan, drag, zoom centré souris standard et zoom centré souris avec grossissement variable selon les composants (zoom différencié).

Documentation :

Toute la doc citée pour le TP1,

Applying Transformations in *JavaFX*
http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/transforms.htm#CHDGCBAH

Exercice 1 : Formes géométriques

1.1 Créez un nouveau projet *JavaFX*, importez et étudiez le code fourni.

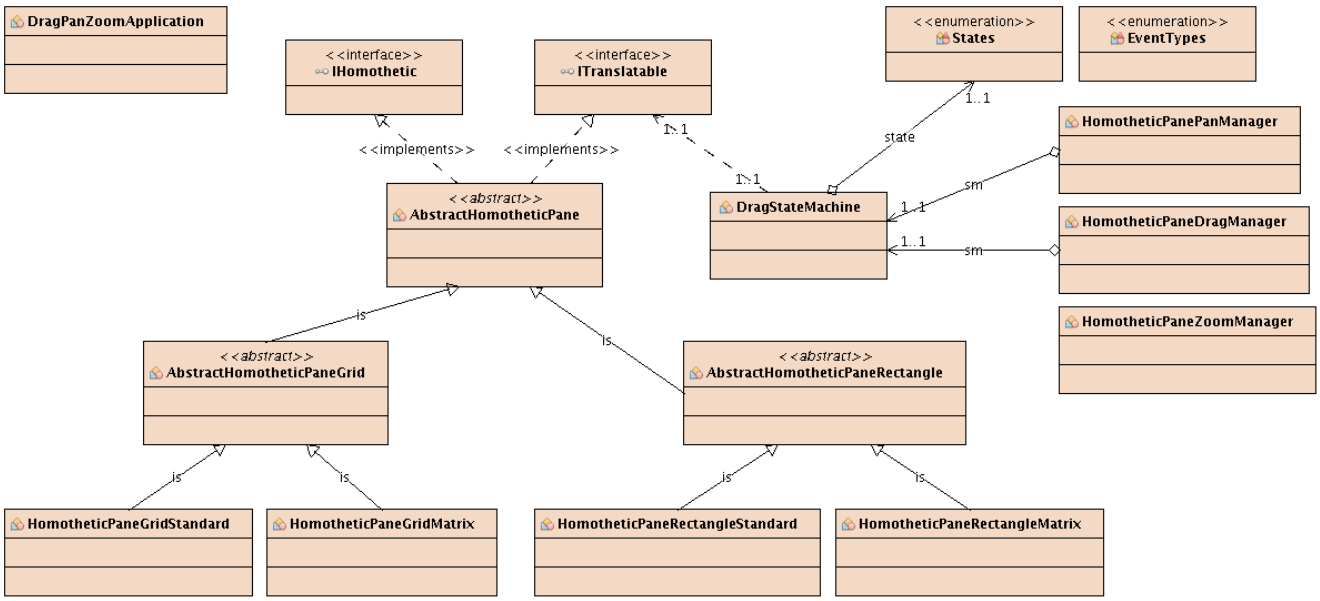
Ce code servira de base pour créer des interactions de manipulation directe d'une ou plusieurs formes dans un conteneur pouvant lui-même être manipulé. Les manipulations seront les suivantes :

- pan du conteneur (translation du conteneur, et donc de tout ce qu'il contient, par manipulation directe avec la souris),
- drag des formes (translation indépendamment du conteneur par manipulation directe avec la souris),
- zoom (du conteneur) centré sur la position pointée par la souris.

L'application aura les spécifications générales suivantes :

- la gestion des interactions devra être mise en place dans les trois classes suivantes en fonction de l'objet géré : `HomotheticPaneZoomManager`, `HomotheticPanePanManager` et `HomotheticPaneDragManager`,
- la gestion du pan et du drag se fera grâce à des machines à états,
- pour avoir un repère lors des transformations, le conteneur devra afficher une grille en arrière-plan,
- deux versions du mécanisme de transformation du conteneur seront implémentées dans deux classes nommées `HomotheticPaneGridStandard` et `HomotheticPaneGridMatrix`,
- les formes à manipuler seront de la même façon des implémentations issues de la classe `AbstractHomotheticPane`.

A la lumière de ces spécifications expliquez les choix qui ont été fait pour l'architecture de cette application. Vous vous aiderez du diagramme de classes ci-dessous présentant schématiquement les classes fournies, qui seront à compléter au cours du TP, ainsi que quelques autres qui restent à créer.



1.2 Dessin de la grille

Quel endroit (quelle classe, quelle méthode) est il le plus adapté au dessin de la grille ?

Dessinez la grille avec un quadrillage de 50px. Vous choisirez la ou les classes d'objets à utiliser dans le package `javafx.scene.shape`. Vous intégrerez cette grille dans un `Group` , lui-même intégré dans le graphe de scène de la classe choisie dans le code fourni. Pour l'intégration inspirez vous de celles qui sont réalisées dans l'application principale.

Y a-t-il un mécanisme particulier à mettre en place pour s'assurer que la grille restera en arrière plan ?

1.3 Schématisez le graphe de scène de l'application.

Dans la suite de ce TP, chaque mention à la grille évoquera l'instance de la classe contenant la grille, par exemple `HomotheticPaneGridStandard`, et non plus les lignes qui constituent la grille elle-même.

Exercice 2 : Pan de la grille

2.1 Positionnement et translation

Chaque nœud du graphe de scène est positionné dans le repère de son parent grâce à deux mécanismes qui font la même chose dans l'absolu (placer le nœud en x,y) mais sont offerts par commodité : l'un est censé s'occuper d'un positionnement supposé absolu (attributs *layoutX* et *layoutY*), et l'autre des translations que l'on voudrait apporter additionnellement (attributs *translateX* et translate*Y*).
Dans `DragPanZoomApplication`, le mécanisme associé au layout est utilisé pour positionner les différents nœuds lors de leur création. Dans les exercices qui suivent vous utiliserez plutôt le mécanisme de translation.

A partir du code fourni et de la javadoc de la classe `Node` identifiez les méthodes associées à ces mécanismes.

2.2 A l'aide de la doc de `MouseEvent` et de la logique décrite ci-après, déterminez quels sont les différents types de cet événement ainsi que les informations convoyées potentiellement utiles pour gérer le pan.

Logique du pan :

- lorsque le bouton de la souris est pressé, sa position P_0 doit être mémorisée,
- lorsque la souris est déplacée vers le point P_1, l'objet écouté doit être translaté d'un vecteur P_0P_1,
- si les coordonnées sont dans le repère de l'objet translaté il n'y a rien d'autre à faire, si elles sont dans le repère de son parent les coordonnées de P_1 doivent alors être mémorisées dans P_0 pour servir de nouvelle base au prochain déplacement de souris.

2.3 Créez les classes `HomotheticPanePanManager` et `DragStateMachine` pour gérer le pan de la grille.
`HomotheticPanePanManager` sera en charge de récupérer les événements *JavaFX* et de les traduire pour la machine à états. Pour les abonnements vous utiliserez des instances de classes héritant de `EventHandler<MouseEvent>` et enregistrerez ces instances comme des filtres sur les différents types de MouseEvent « émis » par la grille. Pour les traductions vous utiliserez l'enum `EventTypes` fournie.

Exercice 3 : Drag du rectangle

3.1 On souhaite maintenant déplacer le rectangle indépendamment de la grille dans laquelle il se trouve.
En considérant que pan et drag sont similaires (translation d'un nœud du graphe par manipulation directe), mettez en place ce dernier dans une nouvelle classe `HomotheticPaneDragManager` capable de gérer le drag d'une réalisation d'`AbstractHomotheticPane` et qui utilisera dans un premier temps une instance de la machine à états réalisée pour le pan de la grille.

3.2 Testez votre drag. Il semble y avoir un conflit entre plusieurs interactions. D'où provient il et comment le gérer ? Décrivez deux solutions distinctes (pensez aux propriétés et au cycle de propagation des événements abordés dans le TP4).

3.3 Implémentez l'une des deux solutions.

3.4 Réalisez maintenant une autre machine à états comprenant une hystérèse.

Exercice 4 : Zoom basique (méthodes standard)

4.1 Identifiez dans la javadoc de la classe `Node` le mécanisme (attributs et méthodes associées) qui permet de modifier l'échelle (scale) d'un nœud. Autour de quel point se fait la mise à l'échelle ?

Pour information il existe d'autres transformations, en 2D et en 3D, mais elles ne seront pas explorées dans le cadre des TP. Les transformations (translations, échelle...) se cumulent à travers le graphe de scène. Ainsi un nœud se voit il appliquer ses propres transformations mais également de proche en proche toutes celles qui ont été appliquées à ses parents jusqu'à la racine de la scène.

4.2 Un squelette de la classe `HomotheticPaneGridStandard` vous est fourni. Dans ce squelette, les méthodes abstraites héritées de `AbstractHomotheticPaneGrid` doivent absolument être implémentées si on veut que l'application compile (vous aviez besoin d'une première version de l'application pour l'exercice 1). Ces méthodes auraient pu être implémentées vides mais il est préférable qu'elles propagent plutôt une exception. Cette exception est elle sous contrôle du compilateur ou pas ? Expliquez la raison de cette façon de faire.

4.3 Implémentez la méthode `setScale(double scale)`. L'écriture est immédiate si vous suivez la logique indiquée dans les commentaires du constructeur.

4.5 Testez le résultat en modifiant l'écouteur des événements clavier dans `DragPanZoomApplication` pour zoomer lorsqu'on appuie sur une touche de votre choix. Quel est le centre de la transformation ainsi réalisée ?

4.6 A l'aide de la documentation de `ScrollEvent` déterminez quel est le type de cet événement ainsi que les informations convoyées potentiellement utiles pour gérer un zoom de la grille par action sur la molette de la souris.

4.7 Créez la classe `HomotheticPaneZoomManager` pour gérer le zoom de la grille.

Avec toujours le même centre de transformation est ce une interaction pratique ? Que pourrait on préférer avoir ?

Exercice 5 : Zoom centré souris (méthodes standard)

5.1 Implémentez la méthode `setScale(double scale, double x, double y)`. L'écriture est plus compliquée que pour `setScale(double scale)` car la mise à l'échelle standard en *JavaFX* se fait par rapport au centre de l'objet. Il vous faudra donc :

- identifier les attributs x et y aux coordonnées d'un point dans le repère local à l'objet,
- calculer et mémoriser les coordonnées de ce point dans le repère de l'écran,
- mettre à l'échelle (par rapport au centre de l'objet), ce qui va modifier les coordonnées du point à l'écran mais pas dans son repère local,
- déterminer les nouvelles coordonnées du point dans le repère de l'écran après mise à l'échelle,
- effectuer la bonne translation pour ramener ce point sous le pointeur souris.

Pour les conversions du repère local au repère de l'écran comme pour la translation vous utiliserez les méthode fournie par la classe `Node`.

5.2 Testez le résultat en modifiant l'écouteur des événements clavier dans `DragPanZoomApplication` pour zoomer par rapport à quelques nœuds arbitraires de la grille de l'exercice 1. Ces nœuds devront rester à leur position après la mise à l'échelle.

5.3 A l'aide de la documentation de `ScrollEvent` déterminez les informations supplémentaires à exploiter pour gérer un zoom centré souris.

5.4 Améliorez le zoom réalisé à l'exercice 4.

Exercice 6 : Zoom différencié (méthodes standard)

6.1 L'utilisation du zoom réalisé précédemment entraîne quel problème de visualisation ?

Quelle solution peut on envisager ?

6.2 Quel type d'application peut il y avoir pour une image radar ?
Considérez pour cela que l'application du TP constitue un modèle graphique et interactif simplifié de l'image radar fournie sous forme de *jar*. Vous pourrez exécuter et explorer le fonctionnement de cette dernière afin d'identifier les correspondances entre les composants visuels des deux applications, ainsi que les différences de traitements qu'ils subissent lorsque vous zoomez.

6.3 Nous allons mettre en place un premier zoom dit différencié, qui aura vocation à être appliqué à la grille. Il fera en sorte que l'épaisseur de ses traits reste visuellement invariante au fur et à mesure des modifications d'échelle.
Attention, toutes les transformations étant cumulées le long du graphe de scène, visuellement invariant ne signifie pas invariant. Cela signifie plutôt qu'il faudra appliquer une transformation inverse aux nœuds dont on souhaite qu'ils ne changent pas d'aspect ou à une propriété des ces nœuds pour qu'elle reste visuellement invariante après le cumul des transformations.

Dans notre cas le traitement ne se fera pas en bloc sur le nœud de la grille (le `Group` contenant la grille dans la classe `AbstractHomotheticPaneGrid`) ni sur les lignes qui composent cette grille (les instances de `Line`) mais sur la propriété *strokeWidth* de chaque ligne, qu'il faudra diviser par la valeur de l'échelle (la propriété *scale* héritée de `AbstractHomotheticPane`) chaque fois que celle-ci change.
Pour écouter les changements de la propriété *scale* il faudra vous abonner grâce à la méthode `addListener(…)` spécifiée dans l'interface `ObservableValue` qu'implémentent toutes les `Property`. Comme écouteur pour cet abonnement vous utiliserez un `ChangeListener<Double>`.
Mettez en place ce mécanisme au plus près de la création de la grille, donc dans la classe `AbstractHomotheticPaneGrid`.

6.4 A quel type de composant visuel de l'image radar devrez vous appliquer le mécanisme mis en place ?

6.5 Grâce à la question 7.2 vous avez dû identifier `AbstractHomotheticPaneRectangle` à une représentation simplifiée de l'étiquette d'un trafic dans l'image radar. L'étiquette nécessite un mécanisme de zoom différencié spécifique puisqu'elle ne doit pas changer de taille à l'écran lorsqu'on zoome.
Implémentez ce mécanisme de zoom différencié.

Cette fois ci la transformation est plus simple puisqu'elle est globale : il suffit d'appliquer l'échelle inverse à l'ensemble du composant lorsque la propriété *scale* de la grille change. Vous aurez donc besoin d'une référence à cette propriété dans la classe.
Pour la mise à l'échelle vous commencerez par utiliser la méthode `setScale(double scale)`. Quel est dans ces conditions le centre de la mise à l'échelle du composant ?

6.6 Mettez maintenant à l'échelle par rapport à l'origine du composant.

En réalité cette solution fonctionne mal car elle repose sur des conversions avant et après la mise à l'échelle. Or le moment de cette mise à l'échelle est optimisé par la machine, selon un mécanisme de layout que l'on ne peut pas contrôler. Du coup il se peut que certaines conversions soient demandées au mauvais moment, ce qui introduit potentiellement des erreurs de placement au fur et à mesure de l'interaction. L'exercice suivant va explorer une autre implémentation corrigeant ce défaut.

Exercice 7 : Zoom différencié (matrices de transformation)

Jusqu'à présent nous avons modifié l'échelle grâce aux méthodes standard de la classe `Node`, en modifiant `scaleXProperty` et `scaleYProperty` (grâce à un binding). Il existe une autre méthode impliquant l'utilisation de matrices de transformations.

7.1 En vous basant sur la doc de la classe `Node`, expliquez ce que permet d'obtenir l'accesseur `getTransforms()`.

7.2 En vous basant sur la javadoc expliquez ce qu'est la classe `Affine` et ce que permet de faire ses méthodes `appendScale(…)`.

7.3 Créez une autre réalisation de la classe abstraite `AbstractHomotheticPaneRectangle`, nommée `HomotheticPaneRectangleMatrix` et chargée d'implémenter la mise à l'échelle avec des matrices de transformation.
Pour ce faire, fonctionnez par analogie avec ce qui a été fait dans la méthode standard :

- la liaison entre le mécanisme de transformation et le modèle réalisée dans le constructeur consistera dorénavant à affecter une transformation affine à la liste des transformations du composant,
- les méthodes `setScale(…)` modifieront simplement cette transformation affine au fur et à mesure des mises à l'échelle.

7.4 Testez votre classe.

La mise à l'échelle par défaut décrite par la méthode `appendScale(double sx, double sy)` utilisée dans vos `setScale(…)` se fait par rapport à l'origine du repère du composant. Vous devriez donc obtenir le même résultat pour le pivot de votre zoom différencié, que vous ayez utilisé `setScale(myScale)` ou `setScale(myScale, 0, 0)`.

7.5 Avec le même mécanisme revenez à la classe abstraite `AbstractHomotheticPaneGrid`, et créez sa réalisation `HomotheticPaneGridMatrix` chargée d'implémenter la mise à l'échelle de la grille avec des matrices de transformation.

Exercice 8 [optionnel] : Alternative aux transformations inverses

8.1 Y a-t-il un autre moyen de faire du zoom différencié ? Pensez en termes de possibilités d'organisation du graphe de scène et de bindings.

8.2 Réalisez le zoom différencié par cet autre moyen.