TP1 : Bases graphisme et interaction

Ce TP aborde les concepts de base de la programmation événementielle. Il sera mené progressivement en même temps que le cours d'introduction à *JavaFX*. **Durée (cours compris) : 4h**

Objectifs

- Assembler les noeuds d'un graphe de scène (formes et widgets) au sein d'un layout simple,
- Mettre en place des transformations géométriques sur ces noeuds,
- Mettre en place une liaison de données,
- Utiliser un modèle de données simple,
- Réagir à des changements de valeur,
- Réagir à un événement tout au long de sa chaîne de propagation.

Documentation

JavaFX User Interface components
http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/index.html

JavaFX Layouts
http://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html

JavaFX Events processing
http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

JavaFX Javadoc

https://docs.oracle.com/javase/8/javafx/api/toc.htm

Exercice 1 : Graphe de scène, nœuds et systèmes de coordonnées

1.1 Lancez *NetBeans* et créez un nouveau projet "TP1" sans classe principale (décochez l'option "Create Application Class"). Copier dans son dossier source la classe `Ex1_CoordinateSystems` fournie avec le sujet sur e-campus.

Cette application met en place des conteneurs imbriqués avec un layout absolu, c'est à dire sans contraintes de placement automatisées. Les enfants des conteneurs sont donc placés explicitement en (x, y) dans leur parent. Une forme rectangulaire est placée à l'intérieur du container le plus bas dans le graphe de scène. Le code permet d'écouter les clics souris sur le rectangle (nous verrons plus tard comment mettre en place cet abonnement, pour l'instant utilisez le juste tel quel) : lorsqu'un clic survient, d'afficher les coordonnées les coordonnées de la souris dans le repère local du rectangle.

Lisez le code, dessinez le graphe de scène pour montrer l'imbrication des conteneurs, et imaginez ce qui sera affiché si on clique dans les coins du rectangle. Vérifiez votre compréhension en exécutant le code.

1.2 Transformez le code de façon à ce qu'il affiche également les coordonnées de la souris dans le repère du parent du rectangle, de la scène et de l'écran. Vous utiliserez pour cela les méthodes de conversion vues en cours (c'est juste pour vous entraîner à les utiliser, vous verrez plus tard que dans ce cas précis il y a mieux à faire).

Exercice 2 : Transformations et systèmes de coordonnées

2.1 Dans le même projet copiez/collez la classe `Ex1_CoordinateSystems` du premier exercice. Ce faisant, l'IDE vous proposera de renommer cette nouvelle classe et vous saisirez `Ex2_CoordinateSystems`. Dans cette nouvelle classe multipliez l'échelle du conteneur `nestedContainer` par 2 (en ajoutant par exemple une nouvelle instance de la classe `Scale` à sa liste des transformations).

2.2 Quel est le résultat de cette modification lorsque vous cliquez dans les coins du rectangle ?

Exercice 3 : Liaison de données

L'un des mécanismes de programmation événementielle de *JavaFX*, la liaison de donnée, permet de lier deux propriétés de façon unidirectionnelle ou bidirectionnelle, assurant que lorsque l'une des deux propriétés est modifiée, sa nouvelle valeur sera automatiquement recopiée dans l'autre. Les propriétés ne peuvent pas être d'un type primitif, elles doivent être d'un sous-type de la classe `Property` qui encapsule un type primitif et lui offre les services d'observabilité rendant possible, entre autres, la liaison de donnée).

3.1 Identifiez dans la javadoc de la classe `Property` (ou dans le cours) les deux méthodes pour mettre en place une liaison de données unidirectionnelle et bidirectionnelle.

3.2 Créez une nouvelle classe principale `Ex3_Bindings` dont la racine du graphe de scène sera un conteneur capable d'aligner ses enfants horizontalement. Placez y deux instances de la classe `Slider` et liez la valeur du second à celle du premier (binding unidirectionnel).

Que se passe-t-il quand on manipule le premier slider ? Même question pour le deuxième.

3.3 Ajoutez une instance de la classe `TextField` entre les deux sliders et liez son contenu à la valeur du premier slider.

Que se passe-t-il ? Quel type de mécanisme faudrait il mettre en place pour que cela marche ?

Cherchez une solution dans la javadoc de la classe `NumberExpressionBase` dont héritent toutes les properties encapsulant un nombre (ou dans le cours…).

3.4 Transformez maintenant la liaison de donnée entre les deux sliders en un binding bidirectionnel.

3.5 Tentez de faire de même entre le contenu du text et la valeur du premier slider. Vous vous heurtez au même problème de conversion de type des données sauf qu'en plus cette fois ci, la conversion doit se faire dans les deux sens. Trouvez la solution dans la doc de `StringProperty` (ou on ne sait jamais, peut-être encore dans dans le cours…).

Exercice 4 : Modèle de données

Les sliders et les texts permettent d'afficher mais aussi de modifier une valeur. Ils seront ainsi qualifiés de ***vue/contrôleur*** et la valeur associée sera qualifiée de ***modèle de données***. La méthode suivie dans l'exercice précédent a l'inconvénient de mêler l'ensemble vue/contrôleur et le modèle de donnée (ici la valeur), ce qui peut vite rendre le code difficile à maintenir.

Créez une copie de la classe `Ex3_Bindings` et nommez la `Ex4_DataModel`. Dans cette nouvelle classe isolez le modèle dans une `Property` adaptée (encapsulant un `Double`) et modifiez toutes les liaisons de donnée pour qu'elles ne se fassent plus entre les différents widgets mais directement au modèle.

Dans les applications plus grosses on pousse la séparation plus loin en externalisant modèles de données et calculs complexes dans des classes spécifiques qui constitueront ce qu'on appelle un ***noyau fonctionnel***. Vous avez déjà manipulé un noyau fonctionnel fourni dans le dernier TP de *POO/Java* (Jeu de Morpion). **Dans votre projet Java** il vous faudra créer ce noyau fonctionnel et le constituer d'objets observables en JavaFX. Vos ***vues/contrôleurs*** n'auront plus alors qu'à observer les parties adaptées du ***modèle*** pour réagir automatiquement.

Exercice 5 : Réagir à un changement de valeur

Pour réagir de façon plus complexe à un changement de valeur (instructions pour un calcul compliqué ou nécessité de créer des effets de bord) on ne peut plus utiliser la liaison de données. A la place il faudrait pouvoir invoquer automatiquement une méthode suivant nos instructions chaque fois qu'il y a un changement de valeur. C'est le rôle du `ChangeListener` et de son mécanisme d'abonnement.

5.1 Créez une copie de la classe `Ex4_DataModel` et nommez la `Ex5_ChangeListener`. Dans cette nouvelle classe ajoutez un `ChangeListener` qui imprime un message d'alerte dans la console chaque fois que le modèle excède une valeur de 75. Attention : il y a des classes nommées

`ChangeListener` dans plusieurs packages *Java*. Prenez bien soin, quand vous utilisez la complétion et l'import, de choisir la bonne option parmi celles que vous propose *NetBeans* (provenant d'un paquetage *JavaFX*, pas d'un paquetage *Swing* ou autre).

Que se passe-t-il lorsque vous bougez continuellement le slider dans la zone au-delà du seuil ?

5.2 On veut maintenant une alerte unique lorsque le seuil est dépassé et une fin d'alerte lorsqu'on redescend en dessous. Chacun de ces messages doit également être daté.

Indication : la date système peut être obtenue et formatée en une chaîne de caractères avec l'instruction `String date = new SimpleDateFormat("HH:mm:ss.SSS").format(new Date());`

5.3 Finalement, parce qu'écrire un message dans la sortie standard n'est pas ce qu'on peut appeler une IHM très avancée, vous allez aussi changer la couleur du champ texte en rouge lors de l'alerte. Evidemment il devra alors redevenir noir en fin d'alerte.

Indications :
Vous pouvez changer la couleur en affectant un style *CSS* au champ texte (la méthode `setStyle(...)`, le style `-fx-text-fill`).

Exercice 6 : Réagir à un événement tout au long de sa chaîne de propagation

Nous avons maintenant besoin de réagir à un événement (par exemple un clic souris) et non plus à un simple changement de valeur. D'une façon similaire à ce que nous avons fait dans l'exercice précédent nous voulons pouvoir invoquer automatiquement une méthode contenant nos instructions pour réagir à un événement, chaque fois que cet événement survient. Ce sera cette fois-ci le rôle de l'`EventListener` et de ses deux mécanismes d'abonnement.

6.1 Récupérez la classe `Ex6_Events` fournie avec le sujet. Elle est basée sur le Helloworld que nous avons étudié en cours. Vous pouvez y voir que le code permet de réagir à l'`ActionEvent` qui est émis lorsque l'utilisateur presse le bouton. L'abonnement est réalisé en utilisant les méthodes de commodité pour un abonnement simplifié, mais dans cet exercice nous voulons étudier la chaîne de propagation des événements, ce qui n'est pas possible avec les méthodes de commodité. Transformez le code pour utiliser l'abonnement complet, avec l'option *filter* ou *handler* à votre convenance. Vérifiez que le code fonctionne toujours.

6.2 Maintenant abonnez vous au même type d'événement mais sur le conteneur à la racine du graphe de scène (donc le parent du bouton), et choisissez l'option qui permettra de réagir avant l'abonnement déjà fait sur le bouton.

6.3 Finalement ajoutez un nouvel abonnement sur la racine du graphe de scène, mais avec l'autre option, permettant ainsi de réagir également après celui sur le bouton.

6.4 Maintenant que se passe-t-il si vous ajoutez l'instruction `event.consume()` à la fin de la méthode `change` écrite à la question 6.2 ?