TP2 : WIMP

Ce TP est dédié au développement d'une application *WIMP* (*Window, Icons, Menus, Pointing device*), c'est à dire une application avec une IHM basée sur des objets graphiques interactifs essentiellement cliquables (icones, menus, boutons…). **Durée : 2h**

Objectifs

- Assembler des nœuds dans un layout riche,
- Réagir à un ensemble varié d'événements,
- Utiliser quelques patrons de conception utiles en IHM,
- Architecturer du code pour séparer le noyau fonctionnel de la visualisation/interaction.

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 : Jeu de Morpion

1.1 Préparation

Lancez *NetBeans* et créer un nouveau projet *Java* (pas *JavaFX*), nommé "TP2" sans classe principale (décochez l'option "Create Application Class"). Copiez le code et les ressources fournis avec le sujet en suivant les indications ci-dessous :

- Coller le contenu du dossier "src" dans le dossier source du projet,
- coller le dossier "resources" à la racine du projet (il contient des images à afficher).

Le code fourni est une correction du jeu de Morpion (Tic-tac-toe) sur laquelle vous avez déjà travaillé en TP *POO/Java*, et comprenant donc maintenant deux implémentations fonctionnelles : l'une en mode textuel et l'autre en *Swing*. Dans ce TP vous allez développer une troisième version, en *JavaFX*. Et vous pourrez le faire facilement car le code est architecturé proprement en suivant un certain nombre de ***patrons de conception*** (ou ***design patterns***).

Après avoir vérifié que l'application fonctionne bien sur votre ordinateur lisez en le code et sa documentation (vous pouvez même générer la javadoc en faisant un clic droit sur le nom du projet et en choisissant l'item "*Generate Javadoc*"). Examinez les diagrammes de classe et de séquence du sujet et répondez aux questions suivantes :

- Quels sont les patrons de conception utilisés ? Il y en a un de plus que dans le code originel.
- Pourquoi sont ils utilisés ?
- Lequel d'entre eux permet de basculer facilement entre chaque implémentation ?
- Lequel d'entre eux facilite un changement éventuel de technologie pour l'IHM (le développement de la version *JavaFX*) sans toucher au noyau fonctionnel ?

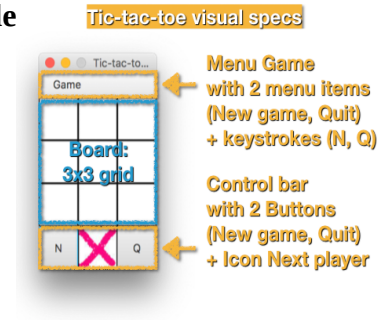
Nous allons maintenant développer la version *JavaFX* en suivant les mêmes étapes que la version *Swing*: utiliser des composants (layouts et widgets) *JavaFX* à la place des composants *Swing*, traduire l'interactivité basique, lier l'ensemble au noyau fonctionnel, et enfin augmenter l'utilisabilité en ajoutant des menus, raccourcis clavier…

Pour vous aider, une classe nommée `TicTacToeJavaFXView` est fournie avec un certain nombre de choses déjà faites. Prenez le temps de la lire pour réviser la structure d'une application *JavaFX* et pour vous rappeler les éléments spécifiques à notre jeu.

1.2 Layout

Le conteneur à la racine du graphe de scène est une `VBox`, c'est à dire un conteneur capable d'aligner ses enfants verticalement conformément à la spécification visuelle ci-dessus. Identifiez dans la doc sur les layouts *JavaFX* les meilleurs candidats pour composer la grille de jeu et la barre de contrôle.

Parce que c'est l'élément le plus simple commencez par la **barre de contrôle** :



- Créez le conteneur qui convient,
- ajoutez le comme enfant de la racine,
- créez les boutons,
- créez l'icone pour afficher le prochain joueur (initialisez la avec l'`Image` nommée `VOID`),
- n'oubliez pas d'ajouter les 3 widgets à leur parent commun.

Pour vous aider, voici une grille de correspondance entre les composants *Swing* et *JavaFX* à utiliser ici :

Swing	***JavaFX***
Fenêtre de l'application : `JFrame`	N'importe quel conteneur peut être mis à la racine du graphe de scène
On choisit un container dont on fixe le layout (par exemple un `Container` avec un `BoxLayout` ou un `JPanel` avec un `GridLayout`)	On choisit directement un Layout Container (`VBox`, `GridPane`, `Hbox`...)
`ImageIcon`	`Image` (attention les chemins d'accès ne s'expriment pas tout à fait de la même façon)
`JLabel`	`ImageView` (seulement pour les images donc ce n'est pas un équivalent strict)
`JButton`	`Button`

Testez votre application (n'oubliez pas de basculer vers la vue *JavaFX* dans la classe principale) et réglez les problèmes d'alignement. La solution la plus simple ici est de donner aux boutons la même taille que l'icone (recherchez la méthode appropriée dans la javadoc de `Button` et d'`Image`).

Passons maintenant à la **grille de jeu** :

- Créez le conteneur qui convient et ajoutez le comme enfant de la racine,
- créez des instances d'`ImageView` pour les cases de la grille...
- mais faites le intelligemment (la grille pourrait bien avoir une taille différente d'un jeu à l'autre),
- ajoutez les cases au tableau `squareViews` (nous aurons besoin d'en garder des références),
- ajoutez les cases à leur parent (il y a un moyen spécifique d'ajouter des éléments à un emplacement spécifié dans une grille, cherchez dans la javadoc).

Nous avons maintenant la même apparence générale que dans la vue *Swing*, à part pour la barre de menu, que nous ajouterons plus tard. Nous pouvons passer à la partie interactivité.

1.3 Interactivité basique et lien au noyau fonctionnel

Commencez par les boutons : révisez le dernier TP et le cours puis choisissez la méthode que vous préférez (abonnement simplifié ou complet, classe interne anonyme implémentant `EventHandler` ou factorisation de cet `EventHandler` dans une instance nommée pour une réutilisation dans tous les boutons et les menus à venir, comme dans la correction *Swing*…). La réaction codée dans la méthode `handle` des `EventHandler` sera souvent la même que dans la version *Swing* puisqu'il s'agira simplement de commander le modèle ou d'appeler une méthode déjà définie.

Passez ensuite aux cases de la grille : comme dans la version Swing factorisez ici la réaction aux clics dans un seul `EventHandler` plutôt que d'en créer un par case. Dans cet unique instance vous récupèrerez la source de l'événement pour identifier la case et retrouver ses indices dans le tableau `squareViews` puis vous signalerez le coup joué au modèle. Conformément au diagramme de

séquence le modèle calculera alors les résultats du coup et appellera les bonnes méthodes de la vue pour la mettre à jour à son tour.

Pour rendre l'ensemble fonctionnel il vous faut enfin implémenter les dernières méthodes vides dans le code fourni (celles qui sont appelées par le modèle. Là encore vous pouvez vous inspirer de la version *Swing* (correspondance des widgets dans le tableau ci-dessous).

Swing	***JavaFX***
`JOptionPane`	`Alert`

Notre *vue JavaFX* est maintenant fonctionnelle. La dernière question est consacrée à l'implémentation de quelques services standards pour améliorer l'utilisabilité de toute application interactive.

1.4 Amélioration de l'utilisabilité

Même si dans notre jeu toutes les actions sont déjà directement accessibles par les cases et les boutons nous allons mettre en place des **menus**. La correspondance des composants est la suivante :

Swing	***JavaFX***
Eléments de menus : `JMenuBar`, `JMenu`, `JMenuItem`	`MenuBar`, `Menu`, `MenuItem`

Les noms des composants se ressemblent beaucoup mais la méthode d'ajout diffère car contrairement à celle de *Swing*, la barre de menu de *JavaFX* est un widget comme les autres, à ajouter à son parent en utilisant les méthodes standards) :

- Créez une instance de la classe `MenuBar`,
- ajoutez la en tant que premier enfant du conteneur à la racine (recherchez dans la javadoc de la classe `java.util.List` pour y insérer un élément à une position spécifiée),
- créez une instance de la classe `Menu` et ajoutez la à la `MenuBar`,
- créez une instance de la classe `MenuItem` et ajoutez la au `Menu`,
- abonnez vous à un `ActionEvent` sur le `MenuItem`.

Si vous aviez déjà factorisé les `EventHandler` pour les boutons, vous pouvez les réutiliser pour les `MenuItem` puisqu'ils tous deux émettent un `ActionEvent`.

Dans toutes les applications, afin d'augmenter l'efficacité des utilisateurs experts, en plus des menus, les plus importantes fonctions devraient toujours être accessibles par des **raccourcis clavier** (en anglais **keyboard shortcuts**). La correspondance des composants est la suivante :

Swing	***JavaFX***
Accélérateurs des items de menus : `KeyStroke`	`KeyCombination`

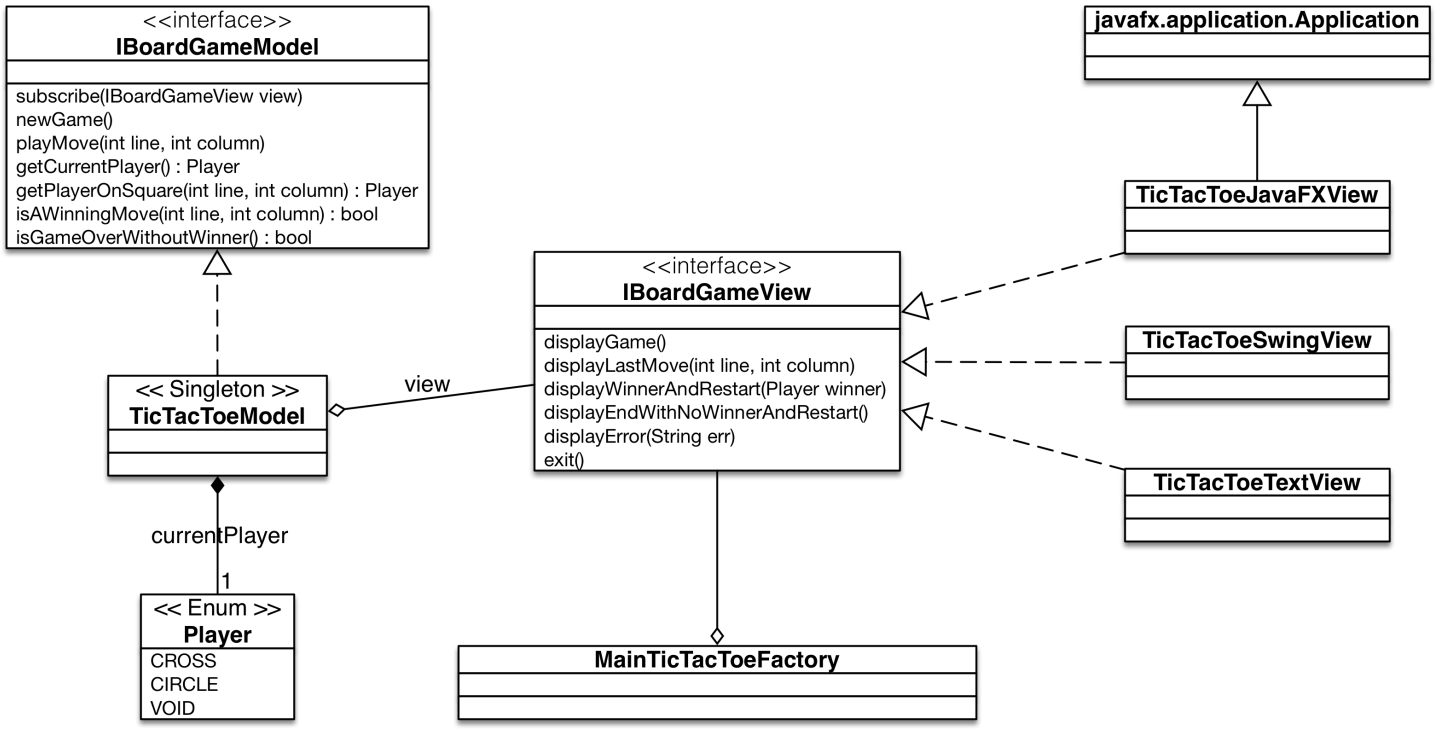
Ici aussi la méthode d'ajout diffère :

- créez une instance de la classe `KeyCombination`,
- associez la à un `MenuItem` grâce à sa méthode `setAccelerator()`.

Il y a encore beaucoup d'améliorations à apporter à notre jeu. Nous n'avons pas le temps de les réaliser mais prenons le temps d'en discuter un peu. Quelles pourraient être ces améliorations ?

Félicitations ! Vous savez maintenant comment développer une application *WIMP* standard. Dans le TP3 vous apprendrez à concevoir et à développer des interactions de *manipulation directe* comme le *pan*, *le drag*, le *zoom centré souris*… **Dans votre projet** vous devrez suivre ces bonnes pratiques.

Annexe 1 : Diagramme de classes



Annexe 2 : Diagramme de séquence