MVC - partie 1

1 Apprentissage

Jusqu'ici, toutes les applications Java-FX que vous avez écrites tiennent en une seule classe, avec parfois un fichier FXML en accompagnement. Le défaut de cette approche, comme nous l'avons vu en cours, c'est que cette classe fourre-tout devient un véritable plat de spaghettis, ce qui complique nettement sa maintenance.

Nous allons donc aujourd'hui étudier le modèle MVC, Modèle Vue Contrôleur, afin de séparer les choses de façon thématique et de rendre les choses plus claires.

1.1 GIT

Reprenez votre dépôt git et créez une branche "tp3" depuis la branche `master`.
Créez-y une nouvelle application Java-FX avec FXML. Prenez l'habitude de bien nommer vos *packages* et vos artéfacts.

1.2 Le fichier FXML

Nous allons commencer par modifier le fichier décrivant l'interface de l'application. Vous pouvez utiliser Scene-Builder si vous le souhaitez.

— Supprimez le contenu de votre scène, et ajoutez-y un **GridPane** de $3x3$ cases.
— Configurez le pour avoir ses lignes visibles, un `padding` de 4, un `hgap` de 1 et un `vgap` de 1.
— Centrez chaque colonne et chaque ligne.
(ce qui nécessite l'ajout de contraintes si vous le faites sans Scene-Builder)
— Donnez lui un `fx:id` égal à "grille".

Voilà tout pour le moment.
Depuis votre EDI, ajoutez à la balise `GridPane` l'attribut "`fx:controller="iut.gon.tp3.GrilleController"`". Utilisez bien sûr le nom de VOTRE *package*...

1.3 Le contrôleur

Lancez votre application. Elle doit planter au lancement, car **FXMLLoader** essaye de créer une instance de la classe **GrilleController**, qui n'existe pas!

Créez donc cette classe. Elle doit bien sûr être publique.
L'application doit maintenant se lancer, et afficher une "belle" grille bien vide. Ce qui est triste.

Ajoutez à votre contrôleur la variable d'instance `private @FXML GridPane grille`.
Cette variable recevra automatiquement le **GridPane** créé par **FXMLLoader** à la lecture du fichier FXML.

Afin de remplir cette grille, nous avons besoin que le contrôleur soit **Initializable**.
Implantez donc cette interface et créez la méthode `initialize(URL,ResourceBundle)` qu'elle décrit.

Cette méthode est exécutée après le chargement du fichier, mais avant le retour de la méthode `load()`.
À des fins de tests, ajoutez lui l'instruction `grille.setStyle("-fx-background-color: seashell");`
Lancez l'application. La grille est maintenant nacrée.

C'est un bon moment four faire votre premier *commit*, s'il n'a pas déjà été fait...

1.4 Le contenu des cases

Mettre le style aurait pu être fait dans le fichier FXML directement... Mais la fonction `initialize` peut faire plus :

— Ajoutez à votre contrôleur un tableau de $3x3$ **Label**.
— À l'aide d'une double-boucle, créez $3x3$ **Label**, que vous ajouterez à la `grille` et au tableau.
— Modifiez leur texte avec un `String.format("L%dC%d", lg, col)`.

Testez votre application. Ça va plus vite qu'en FXML, n'est-ce pas ?

Ajoutez à chaque **Label** un `onMouseClicked` qui modifie son texte en "bonjour", et testez.
Évidemment, il faut cliquer exactement sur le label pour que ça fonctionne...

Modifiez donc la taille maximale de vos **Label** pour qu'ils remplissent toute la case (par exemple $max = 1000$).
C'est mieux, non ? Avec un `alignment` centré, c'est encore meilleur !
→ Attention, la propriété `alignment` gère la position du bloc de texte, alors que `textAlignment` gère la position des lignes DANS le bloc de texte.

Committez vos modifications.

1.5 Le modèle

Vous avez bien travaillé sur la partie graphique. On va maintenant s'intéresser à la partie modèle...

— Créez une classe **GrilleModel** dans votre *package*.
— Ajoutez-lui un tableau de $3x3$ **String**, que vous remplirez comme vous le souhaitez.
— Ajoutez également les méthodes `getCase(lg,col)` et un `setCase(lg,col,texte)` qui consultent ou modifient respectivement une case du tableau.

1.6 Afficher le modèle

Afin que le fichier FXML puisse afficher le contenu du modèle, la vue doit connaître le modèle. (On a vu en cours que l'inverse était mal...)

Ici, le plus simple est de créer le contrôleur nous-même et de lui passer le modèle avant de charger le fichier FXML :

— Ajoutez un constructeur à la classe **GrilleController** qui reçoive un **GrilleModel** en paramètre.
— Modifiez la fonction `initialize` pour utiliser le contenu du modèle pour remplir les cases.
— Dans l'application principale (fonction `start`), créez un **GrilleModel**.
— Créez ensuite un **GrilleController** en lui passant ce modèle.
— Passez ce contrôleur au **FXMLLoader** (méthode `setController`).
— Laissez ensuite la scène être créée avec la fonction `load()`.
— N'oubliez pas de retirer le contrôleur du fichier FXML.

Testez votre application. La grille présente maintenant les données du modèle, mais répond toujours au clic.
Pensez à modifier le modèle au clic d'un **Label**, pour assurer la cohérence...

Committez vos fichiers.

1.7 Le contrôleur clavier

Dans la fonction `start`, ajoutez à la fenêtre un *event-handler* pour l'événement **KeyEvent.KEY_PRESSED** :

```
stage.addEventHandler(KeyEvent.KEY_PRESSED, event -> {
      switch (event.getText()) {
        case "1" : modele.setCase(2,0, "Touche"); break;// en bas à gauche
      }
    });
```

Que fait ce code ? Complétez le pour les touches de 1 à 9.

Testez votre application. Pourquoi ne se passe-t-il rien ?
-> La vue n'est évidemment pas au courant de la modification, et ne change donc pas son affichage !
Il faudrait donc demander à la vue de se rafraîchir ! Nous allons cependant faire autrement...

1.8 Un modèle observable

Une autre solution, vue en cours, serait de rendre le modèle observable. L'observateur, patron de conception que nous verrons l'an prochain, permet en effet à un *sujet* de *notifier* une demande de *mise à jour* à ses *observateurs*.

Java-FX propose des observables "naturels" : les propriétés.

- Modifiez la classe **GrilleModel** pour stocker un tableau de **SimpleStringProperty**
- Le *getter* retournera la propriété
- Le *setter* modifiera la valeur contenue DANS la propriété
- Modifiez la méthode `initialize` du **GrilleController** pour lier la propriété `text` de chaque **Label** à la propriété correspondante du modèle.

Testez votre application, et committez vos fichiers.

2 Réutilisation

Revenez à la branche "gribouille_stable" et créez une nouvelle branche "gribouille_tp3".
Vous pouvez utiliser des branches temporaires à chaque question si vous souhaiter *commiter* des modifications non testées.
Essayez toutefois de ne *commiter* que des version testées sur la branche principale...

2.1 Finalisation du FXML

Donnez des `fx:id` aux différents composants de l'interface avec lesquels on va interagir :
Pane, **Labels** (pas les fixes), **Canvas**, **ColorPicker**, **Rectangle** (colorés).

Depuis Scene-Builder, dans le menu `View`, activez la fonction "Show Sample Controller Skeleton".
Scene-Builder vous fabrique alors un contrôleur type que vous pouvez copier-coller dans votre EDI.

- Ajoutez votre contrôleur au fichier FXML.
- Rendez votre contrôleur **Initializable**.
- Dans la fonction `initialize`, liez la largeur et la hauteur du **Canvas** à celles du **Pane**. Le **Canvas** n'étant pas naturellement redimensionnable, cela le forcera à occuper l'espace disponible dans la fenêtre.

2.2 Le dessin

On va déplacer les événements créés au milieu de la fonction `start` vers des fonctions de premier plan :

— Dans votre contrôleur, créez deux variables d'instance privées **double** `prevX` et `prevY`.
— Il est inutile d'extraire le **Canvas** de la scène, **FXMLLoader** l'injecte directement dans les variables du contrôleur portant le même nom que les `fx:id`.
— Ajoutez une fonction `public void onMousePressed(MouseEvent evt).`
Cette fonction recevra le contenu du premier événement souris.
— Dans votre FXML, ajoutez un attribut "`onMousePressed="#onMousePressed"`" à votre **Canvas**.
Cette fonction sera appelée automatiquement dès que la souris est enfoncée sur la zone de dessin.
— Faites de même avec la fonction `onMouseDragged`, qui sera appelée lorsque la souris bouge avec le bouton enfoncé.
Cette fonction contiendra le deuxième événement souris.

Testez votre application et essayez de dessiner.
Vous noterez que l'on ne peut plus sortir du **Canvas**...

Commitez vos fichiers.

2.3 Le modèle

Si vous réduisez la fenêtre et que vous l'agrandissez, vous verrez que des portions du dessin sont perdues.
La mémoire du **Canvas** se limite en effet à la zone affichée.

— Ajoutez à votre projet le *package* `modele` que vous trouverez sur eCampus.
Il contient plusieurs classes destinées à mémoriser les dessins.
— Créez un **Dessin** dans la méthode principale (`start`) et passez-le au contrôleur FXML.
— Modifiez les événements souris pour qu'ils créent et modifient des **Traces**, que vous stockerez dans votre **Dessin**.
— Ajoutez un écouteur aux propriétés `width` et `height` du **Canvas**.
Dans cet écouteur, vous parcourrez la liste des **Figure** du **Dessin**, et vous (re)dessinerez chacune d'elle sur le **Canvas**.

Testez votre application, et vérifiez que le dessin se redessine bien lorsque la fenêtre s'agrandit.

Commitez vos fichiers.

2.4 Les liaisons (dangereuses ?)

— La classe **Dessin** contient une propriété `nomDuFichier`.
Liez le titre de la fenêtre à celle-ci.
— Liez vos variables `prevX` et `prevY` aux **Label** de la barre d'état correspondants.
Vous devrez bien sûr les remplacer par des **SimpleDoubleProperty**...
Vous pouvez aussi définir une méthode `onMouseMoved` si vous le souhaitez...

Testez votre application et *committez* la version finale.

3 Rendu

N'oubliez pas de commiter vos fichiers régulièrement, de fusionner les nouveautés dans la branche "gribouille_stable" avec un *squash* et de faire un *push* des branches à la fin !