TP 6 PRGA : IHM JavaFX et B.D. pour Mots Croisés
L3 Miage – 2022-2023
à rendre pour le lundi 3 avril

Le but de ce TP est de réaliser une interface JavaFX pour des grilles de mots croisés stockées dans les tables TP6_GRILLE et TP6_MOT de la base MySQL *base_bousse*. Vous vous appuierez à cet effet sur la classe *MotsCroises* du TP1, en ajoutant à celle-ci des propriétés observables, suivant la démarche déjà pratiquée lors du TP4.

A l'issue des deux étapes de développement que comporte ce sujet, votre application devra proposer les fonctionnalités suivantes :

1. au démarrage : chargement d'une grille prise aléatoirement dans la B.D. ;
2. affichage d'une grille aux bonnes dimensions, présentant des cases blanches vides et des cases noires aux emplacements voulus ;
3. l'utilisateur saisit directement ses propositions dans les cases blanches, après sélection de l'une d'elles.

Les deux étapes de développement sont respectivement : IHM pour grille à dimensions fixes et IHM pour grille à dimensions variables extraite d'une BD.

1. Première étape : IHM pour grille à dimensions fixes

Dans cette première version vous réaliserez une IHM pour une grille unique et imposée de taille 2x3, tout en développant des classes qui pourront facilement être adaptées à la version suivante. Une méthode permettant de créer une telle grille est fournie dans la classe *MotsCroisesFactory* jointe à ce sujet.

1.1 Préparation d'un modèle « JavaFX ready »

Afin de faciliter le chargement puis l'observation de à observer par les classes de l'IHM, vous apporterez les modifications suivantes à l'une des classes *MotsCroises* (la vôtre ou la classe *MotsCroisesCorrigeTP1* fournie, au choix), le résultat étant une nouvelle classe *MotsCroisesTP6* qui constituera le modèle « JavaFX ready » de votre application :

- modifier le type des propositions en remplaçant *Character* par *StringProperty* ; initialiser chaque proposition à partir d'une nouvelle instance de *SimpleStringProperty* contenant la chaîne " " (caractère espace) ;
- toujours dans le constructeur, « noircir » toutes les cases afin de faciliter le chargement ultérieur depuis la B.D. ;
- corriger les méthodes *getProposition()* et s*etProposition()* en conséquence : *getProposition()* doit retourner le premier caractère de la valeur contenue dans la *StringProperty* de la case visée, tandis que s*etProposition()* enregistre dans cette même propriété une chaîne créée à partir de l'unique caractère donné en paramètre ;
- ajoutez à votre classe *MotsCroisesTP6* une méthode *propositionProperty(int lig, int col)* retournant la *StringProperty* située dans la case indiquée, au lieu de la valeur encapsulée dans celle-ci.

En vue de l'étape 1.5, il vous faudra aussi écrire la méthode :

```
public void montrerSolution(int lig, int col)
```

...chargée de remplacer le *char* de la proposition par celui de la solution, dans la case indiquée.

1.2 Création d'une IHM JavaFX minimale

Comme lors du TP4, un fichier « vue » vous est fourni à l'avance : ***VueTP6.fxml***. En l'ouvrant par *SceneBuilder*, vous remarquerez qu'il se limite à un *BorderPane* contenant un unique *GridPane*, lui-même contenant 6 *TextField* disposés sur 2 lignes de 3 colonnes.

Seul le *GridPane* est nommé par une propriété *fx:id* (cf. volet *Code* dans le panneau droit du *SceneBuilder*) : il vous faudra associer cet objet à une variable Java déclarée dans votre classe de

contrôleur, ayant le même nom et le même type que son correspondant XML. Cette variable n'a pas besoin d'être initialisée à ce stade car c'est le *FXMLLoader* de la classe *MainTP6* fournie qui s'en charge, d'où la nécessité de faire précéder sa déclaration par une annotation *@FXML*

Par la suite la plus grande partie du code Java restant à écrire se trouvera dans la méthode *initialize()* du contrôleur, celle-ci devant également être précédée de l'annotation *@FXML*. Vous commencerez par y créer une instance de *MotsCroisesTP6* aux bonnes dimensions, à l'aide de la méthode *creerMotsCroises2x3()* de la classe *MotsCroisesFactory* fournie.

La stratégie de représentation du modèle est la suivante : chaque case blanche est représentée par un *TextField* dotée des propriétés *GridPane.columnIndex* et *GridPane.rowIndex* appropriées (cf. code source du fichier *.fxml*, visible directement dans Eclipse). La base de numérotation étant 0, un *TextField* situé en colonne 1 et ligne 2 sera déclaré comme suit en XML :

```
<TextField ... GridPane.columnIndex="0" GridPane.rowIndex="1" />
```

Dans le code fourni, la solution choisie pour représenter les cases noires est de ne pas les représenter : le *GridPane* étant doté d'un fond noir (cf. sa propriété *fx-background-color)*, si l'on veut laisser une case noire au milieu de la 2ème ligne par exemple il suffira de ne déclarer aucun *TextField* ayant pour propriétés `GridPane.columnIndex="1"` et `GridPane.rowIndex="1"` .

Les trois sections suivantes vous permettront de rendre votre grille interactive, en permettant la saisie des propositions (1.3), en affichant les définitions sous la forme d'infobulles (1.4), et en révélant les solutions sur demande (1.5). Il vous faudra à chaque fois modifier en conséquence les propriétés de chaque *TextField* du *GridPane*, chargé de représenter une case du modèle. Ces instructions devront être placées à l'intérieur d'une boucle parcourant les « enfants » du *GridPane*, comme suit :

```
for (Node n : grilleMC.getChildren()) {
   if (n instanceof TextField) {
      TextField tf = (TextField) n ;
      int lig = ((int) n.getProperties().get("gridpane-row")) + 1 ;
      int col = ((int) n.getProperties().get("gridpane-column")) + 1 ;
      // Initialisation du TextField tf ayant pour coordonnées (lig, col)
      // (cf. sections 1.3, 1.4 et 1.5)
   }
}
```

1.3 Saisie des propositions
Vous établirez à cet effet un lien bidirectionnel entre les propositions connues par le modèle et celles saisies et/ou affichées dans les *TextField* correspondants. Ainsi non seulement les *StringProperty* du modèle pourront « observer » les propositions saisies dans les TextField correspondants, et se mettre à jour en conséquence (ce qui est le but de cette étape) mais en plus on bénéficiera d'un lien réciproque permettant de bénéficier d'une mise à jour automatique d'un *TextField* suite à une révélation de solution effectuée par le modèle à la demande du joueur (but de l'étape 1.5).

Il vous faudra pour cela invoquer *bindBidirectional()* sur la propriété « texte » de chaque TextField, celle-ci étant retournée par la méthode *textProperty()* de cette classe. Le paramètre du « bind » bidirectionnel sera la propriété correspondante du modèle, rendue accessible par une des méthodes réalisées à l'étape 1.

Vu que cette opération exige d'avoir d'une part un pointeur sur chaque *TextField*, et d'autre part les coordonnées (ligne, colonne) de la case de modèle concernée, le code qui la met en œuvre devra logiquement être placé dans la séquence *Initialisation du TextField etc.* donnée ci-haut.

1.4 Affichage des infobulles
Il suffit pour cela d'ajouter cette instruction dans la séquence *Initialisation du TextField etc.* :

```
(référence sur le TextField).setTooltip(new Tooltip(texte)) ;
```

... où « texte » est soit null, soit une chaîne donnant la définition horizontale, la définition verticale, ou les deux avec un « / » de séparation. Quelques exemples tirés de la grille n°10 de la B.D. :

- ligne 3, colonne 1 (horizontal) : *Mois*
- ligne 1, colonne 3 (vertical): *Chauffé*
- ligne 4, colonne 5 (horizontal / vertical) : *Version originale / boissons*

1.5 « Révélation » d'une solution sur demande

Vous commencerez par programmer la réaction au clic de souris sur une case donnée, réalisée par cette méthode de votre contrôleur :

```
@FXML
public void clicLettre(MouseEvent e) {
      if (e.getButton() == MouseButton.MIDDLE) {
            // C'est un clic "central"
            TextField case = (TextField) e.getSource();
            int lig = // n° ligne de la case (cf. boucle du 1.2)
            int col = // n° colonne de la case (cf. boucle du 1.2)
            // appel de montrerSolution() sur le modèle
      }
}
```

Il vous faudra ensuite enregistrer cette méthode en tant que « listener » auprès de chaque *TextField* du *GridPane* de votre vue. Cette manip peut être réalisée de deux manières :

- l'approche « simplifiée » (mais plutôt fastidieuse) consiste à associer dans *VueTP6.fxml* la méthode *clicLettre()* à chacun des cinq *TextField* enfants du *GridPane* :
  - sélectionner d'abord le *TextField,* soit directement dans le panneau central du *SceneBuilder*, soit dans l'avant-dernier volet de nom *Hierarchy* du panneau gauche ;
  - ensuite dans le volet *Code* situé tout en bas du panneau droit, sélectionner *clicLettre()* dans la liste des choix proposés pour *onMouseClicked()*. Si la méthode n'apparaît pas, c'est soit parce que votre classe contrôleur n'a pas été associée à la vue (*Controller* en bas à gauche du SceneBuilder), soit parce que la méthode *clicLettre()* n'a pas été déclarée correctement, et/ou précédée de l'annotation *@FXML*

- l'approche « programmée » consiste à écrire une lambda-expression appelant votre méthode, et à affecter celle-ci à chacun des « enfants » du *GridPane*, toujours depuis la séquence *Initialisation du TextField etc.* de votre contrôleur :

  ```
  tf.setOnMouseClicked((e)->{ this.clicLettre(e);}) ;
  ```

2. Deuxième étape : IHM pour grille variable lue depuis une BD

Chargement d'une grille depuis les tables *TP5_GRILLE* et *TP5_MOT*

Vous ferez en sorte d'appeler la méthode *extraireGrille()* (cf. TP4) au début de la méthode *initialize()* de votre contrôleur. Cependant celle-ci devra dans un premier temps supprimer tous les enfants *TextField* du *GridPane*, et dans un deuxième temps, recréer autant de nouveaux *TextField* qu'il y a de cases blanches (on pourrait bien sûr se contenter de supprimer les *TextField* correspondant aux cases noires parmi les 6 déjà présentes, mais cela reviendrait à interdire les grilles de dimensions différentes)

Voici la marche à suivre pour obtenir ce résultat (il est conseillé d'isoler cette séquence dans une méthode *private* du contrôleur) :

- sauvegarder dans une variable locale *modeleTF* le premier *TextField*, donné par cette expression : `(TextField) grilleMC.getChildren().get(0)`
- supprimer tous les *TextField* contenus dans le *GridPane* : `grilleMC.getChildren().clear();`
- pour chaque case blanche du modèle (c'est-à-dire l'instance de *MotsCroisesTP6* retourné par *extraireGrille*), réaliser les opérations suivantes :

- créer une nouvelle instance de *TextField*
- affecter au nouveau *TextField* les dimensions du *TextField* sauvegardé, par des appels de *setPrefWidth(modeleTF.getPrefWidth())* puis de *setPrefHeight(modeleTF.getPrefHeight())*
- affecter au nouveau *TextField* toutes les autres propriétés du *TextField* sauvegardé, suivant cet exemple :

```
for (Object cle : modeleTF.getProperties().keySet())
{
      nouveau.getProperties()
            .put(cle, modeleTF.getProperties().get(cle)) ;
}
```

- ne pas oublier d'ajouter le nouveau *TextField* au *GridPane* :
`grilleMC.add(nouveau,`*`indice_colonne`*`,`*`indice_ligne`*`) ;`
(attention à la borne inférieure : 0 et non 1 !)

3. Troisième étape : grille renouvelée aléatoirement (et autres améliorations)

Afin de pouvoir proposer de nouvelles grilles à l'utilisateur, la solution la plus simple consiste à ajouter à la vue existante un bouton « nouvelle grille » déclenchant la séquence suivante :

- sélection aléatoire d'une grille dans la base de données ;
- appel de la méthode *extraireGrille()* de l'étape 2, qu'il faudra au passage doter d'un paramètre « n° grille » permettant d'obtenir l'extraction de la grille sélectionnée ;
- chargement de la grille dans la vue, en appelant la méthode *private* également développée à l'étape 2

Une solution plus avancée consiste à développer une deuxième vue de type *.fxml* dotée d'un menu proposant des choix tels que « grille aléatoire », « choix de la grille », « quitter », etc. Cette vue deviendra en fait la vue principale de votre application ; bien entendu elle devra avoir son contrôleur associé, sous la forme d'une nouvelle classe java dotée d'un *initialize(),* etc.

Au delà de ces fonctionnalités minimales, vous réaliserez celles-ci :

- avoir une case courante, avec le focus de l'application (encadré en bleu sur l'image, en utilisant une feuille de style CSS) ;
- conserver une direction courante (horizontale à droite ou verticale vers le bas seulement) ;

Plus précisément :

- un clic de la souris sur une case blanche transforme cette case en case courante ;
- un clic de la souris sur une case noire ne doit rien modifier ;
- une frappe de touche entre "A" et "Z" (majuscule ou minsucule) affiche la lettre dans la case courante et avance la case courante dans la direction courante (à droite ou en bas), s'il existe une telle case (blanche) ;
- il est évidemment impossible de rentrer plusieurs lettres dans une case.

Les fonctionnalités suivantes seront notées avec des point « bonus » :

- la frappe d'une touche fléchée déplace la case courante sur la case suivante dans la direction indiquée, celle-ci étant conservée ;
- la frappe de la touche "Entrée" colorie en vert le fond de chacune des cases dans lesquelles la lettre proposée coïncide avec la solution (bien sûr, la couleur verte doit disparaître si une nouvelle lettre est saisie dans cette case plus tard) ;
- la lettre tapée doit apparaître en s'agrandissant et en majuscule, sans l'accent (utiliser la classe *ScaleTransition* en mettant un *ChangeListener* à *square.textProperty()*) ;
- la frappe de la touche "effacer" (KeyCode.BACK_SPACE) efface la lettre de la case courante, tandis que le curseur se déplace dans la direction opposée à la direction courante (autant que possible bien sûr) ;