Gestion des événements

1 Apprentissage

1.1 GIT

Reprenez votre dépôt git et créez une branche “tp2” depuis la branche `master`. Ajoutez-y le contenu de l’archive que vous trouverez sur eCampus.

L’archive que vous avez téléchargée contient une application incomplète dotée d’une seule fenêtre. Vous devriez pouvoir la compiler et l’exécuter directement.

La fonction `extraitIds` est indigeste, mais on fera mieux plus tard, avec un contrôleur JavaFX…

1.2 Les menus

Vous pouvez observer dans le fichier `fxml` que la barre de menus est vide. Vous allez donc la remplir dans la fonction `prepareMenus`, qui reçoit en paramètre l’objet JavaFX correspondant à la barre de menu à modifier. On procédera en deux étapes : créer les menus puis créer leurs gestionnaires d’événements.

1.2.1 Création des menus

Une **MenuBar**, telle que définie dans le fichier `fxml`, contient des objets **Menu**.

— Créez deux **Menu** avec “_Fichiers” et “_Aide” comme texte.
— Ajoutez-les à la barre de menus grâce à la méthode `addAll` de sa liste observable `getMenus()`.
Comme la liste est observable, la barre de menus sait que ses menus ont changé et qu’elle doit se redessiner…
— Testez l’application. Appuyez sur la touche “Alt”. À quoi servent les caractères ‘_’ ? (un indice?)
— Créez deux **MenuItem** “Quitter” et “À propos”.
— Insérez-les respectivement dans les deux menus que vous avez créés précédemment (via leur propriété `getItems()`).

1.2.2 Création des événements

Comme beaucoup de contrôles JavaFX, les **MenuItem** répondent à l’événement `Action`. Vous pouvez intercepter cet événement avec trois méthodes :
addEventFilter(ActionEvent.ACTION, gestion) sur la **MenuBar**, le **BorderPane**, la **Scene** ou le **Stage**.
addEventHandler(ActionEvent.ACTION, gestion) sur le **MenuItem**, le **Menu** ou les mêmes que ci-dessus.
setOnAction(gestion) sur le **MenuItem** ou sur le **Menu**.

Je vous rappelle que les *filters* sont d’abord exécutés de la fenêtre vers le **MenuItem**, puis les *handlers* du **MenuItem** à la fenêtre. Le gestionnaire du `setOnAction` est exécuté après les *handlers* des composants et est unique (alors qu’il peut y avoir plusieurs *handlers* et *filters*) sur le même contrôle.

Interceptez donc l’événement `Action` sur chacun des **MenuItem** :
Quitter fermera l’application grâce à l’appel `Platform.exit()`.
À propos ouvrira une boîte de dialogue :

— Créez un **Alert** avec un type `NONE` et un `ButtonType.CLOSE`
— Réglez son titre et son contenu (“Fait par…vous!”)
— Appelez sa méthode `show()`

N’oubliez pas de commiter vos fichiers sur le dépôt GIT…

1.3 Les listes

Les listes de la fenêtre sont actuellement vides. Vous allez y remédier avec la méthode `prepareListe`.

Commencez par ajouter des **String** de votre choix à la liste `getItems()` de la liste `gauche`. Testez votre application.

Maintenant que la liste est non-vide, on va pouvoir déplacer les éléments d'une liste à l'autre.
Activez donc le bouton `ajouteTout` en positionnant sa propriété `disable` à *false*.

1.4 Les boutons

Chaque bouton répond à l'événement `Action`, comme les menus.

1.4.1 Ajouter tout / Retirer tout

Les actions des boutons `ajouteTout` et `retireTout` sont déjà interceptées par les méthodes `onAjouteTout` et `onRetireTout`. Le transfert des données d'une liste à l'autre est déjà implanté pour l'axe gauche → droite.

Complétez le transfert des données pour l'axe droite → gauche et activez/désactivez les deux boutons.

1.4.2 Ajouter/Retirer un élément

Restent maintenant les deux boutons centraux. On laissera le problème d'activer/désactiver les boutons en fonction de la sélection pour plus tard, une fois qu'on aura vu le *databinding*.

La sélection d'une **ListView** repose sur un **SelectionModel** qui propose deux méthodes `getSelectedIndex()` et `clearSelection()`.

- Interceptez l'événement `Action` du bouton `versDroite`.
- Récupérez l'index sélectionné de la liste `gauche` (méthode `getSelectedIndex()`).
- S'il est $\neq -1$, retirez l'élément de la liste de gauche (méthode `remove(index)`) et ajoutez-le à la liste de droite (méthode `add(élément)`).
- Annulez la sélection avec la méthode `clearSelection()`.
- Testez votre application.
- Dupliquez le processus pour le bouton `versGauche`.
- Testez le déplacement des éléments d'une liste à l'autre et réciproquement.

N'oubliez pas de commiter vos fichiers sur le dépôt GIT…

1.5 La fenêtre

Stage, qui représente une fenêtre, répond lui-aussi à divers événements. Par exemple, `WINDOW_CLOSE_REQUEST` se produit lorsque l'on clique sur la croix de la fenêtre, avant la fermeture de la fenêtre...

La méthode `prepareFermeture` intercepte cet événement. Complétez cette méthode pour demander confirmation à l'utilisateur :

- Créez un **Alert** de type `CONFIRMATION`, avec deux boutons de types `YES` et `NO`.
- Réglez les paramètres de la boîte de dialogue à votre convenance.
- Appelez sa méthode `showAndWait`.
  Contrairement à `show`, cette méthode est bloquante et retourne un résultat.
  Cet **Optional<ButtonType>** peut être présent (si l'on clique un bouton) ou absent (si l'on ferme le dialogue). S'il est présent, sa méthode `get()` récupère le type du bouton choisi.
  La méthode `orElse(defaut)` retourne le contenu de l'**Optional** s'il est présent ou la valeur par défaut sinon.
  La méthode `orElseGet(supplier)` fait de même, mais avec une fonction qui calculera la valeur par défaut.
- Si l'utilisateur n'a pas répondu "Oui", alors appeler la méthode `consume()` de l'événement de la fenêtre. Celle-ci empêchera l'événement de continuer à se propager, et donc de fermer la fenêtre...

Testez votre application.
N'oubliez pas de commiter vos fichiers sur le GIT.

2 Réutilisation

Revenez à la branche "gribouille_stable" et créez une nouvelle branche "gribouille_tp2". Pensez à rentrer dans cette branche. (vérifiez!)

2.1 Fermeture de la fenêtre

Interceptez l'événement de demande de fermeture de la fenêtre et demandez-en confirmation :

- Créez une classe **Dialogues** dans laquelle vous créerez une méthode statique `boolean confirmation()`.
- Dans cette méthode ouvrez une boîte de dialogue comme ci-dessus et retournez un **boolean** vrai si l'utilisateur a répondu "oui", et faux sinon.
- Dans la méthode `start`, interceptez l'évènement de demande de fermeture de la fenêtre, appelez votre méthode ci-dessus, et consommez l'événement si l'utilisateur n'a pas répondu "oui".

Testez votre application, et *commitez* vos fichiers.

2.2 Dessin

L'objectif de l'application est de permettre à l'utilisateur de dessiner à main levée, comme sur une ardoise.
Le **Canvas** permet justement de servir d'ardoise. Il reste donc à ajouter les événements permettant de faire le travail...

- Créez une branche "gribouille_dessin"
- Créez deux variables d'instance privées **double** `prevX` et `prevY`.
- Extrayez le **Canvas** de la scène avec :
  ```
  Canvas dessin = (Canvas) scene.lookup("Canvas");
  ```
- Ajoutez un *event-handler* à votre **Canvas** pour l'événement **MouseEvent.MOUSE_PRESSED**.
  Cette fonction sera appelée automatiquement dès que la souris est enfoncée sur la zone de dessin.
  Cette fonction copiera simplement les coordonnées X et Y de l'événement dans les deux variables d'instance.

— Faites de même avec l'événement **MouseEvent.MOUSE_DRAGGED**, qui sera appelée lorsque la souris bouge avec le bouton enfoncé. Cette fonction fera :
`dessin.getGraphicsContext2D().strokeLine(prevX, prevY, evt.getX(), evt.getY());`
(La méthode `strokeLine` trace un trait entre les deux points (X, Y) spécifiés.)

Testez votre application et essayez de dessiner... C'est presque ça!
-> Le problème vient du fait que l'on garde toujours le même point de départ pour chacun des segments.
Corrigez et retestez!

Si l'on sort du **Canvas**, le dessin est tronqué, mais on corrigera cela la semaine prochaine.

Commitez vos fichiers.

2.3 Un peu de géométrie

— Créez une branche "gribouille_geometrie"
— Extrayez le **Pane** de la scène avec :
`Pane pane = (Pane) dessin.getParent();`
— Ajoutez un *event-filter* à votre **Pane** pour l'événement **MouseEvent.MOUSE_PRESSED**.
Cette fonction sera appelée automatiquement dès que la souris est enfoncée sur le panneau qui contient la zone de dessin.
— Dans cet événement, ajoutez aux enfants (`getChildren()`) du **Pane** un **Circle** de 5 pixels de rayon autour de la souris si le bouton droit est enfoncé (`event.getButton() == MouseButton.SECONDARY`).

Testez votre application et essayez de dessiner avec le bouton droit.
Quelle différence si on ajoute un `event.consume()` dans la conditionnelle?

Que se passe-t-il si vous commencez un dessin directement sur un cercle?
-> Ça ne fonctionne pas car le cercle reçoit les événements de glissé de souris.

Ajoutez un appel à `setMouseTransparent(true)` sur votre cercle. Cela le rend "transparent" à la souris.
Retestez. -> ça marche!

Commitez vos fichiers, faites un *push*.

2.4 Fusion

Revenez à la branche "gribouille_tp2" et fusionnez la branche "gribouille_dessin".
On laissera de côté la branche "géométrie" pour le moment.

Fusionnez la branche "gribouille_tp2" dans la branche "gribouille_stable". (Attention à l'ordre!)
`git merge --squash gribouille_tp2`.

Quelle est la différence avec un *merge* classique?
→ L'intérêt est de conserver une branche stable "propre", avec un *commit* par fonctionnalité.
→ L'inconvénient est que l'on perd la "parenté" avec la branche fusionnée.

3 Rendu

N'oubliez pas de faire un *push* de chaque branche pour envoyer les commits au serveur!