JavaFX 8
Exercices

zenika
ARCHITECTURE INFORMATIQUE

Table des matières

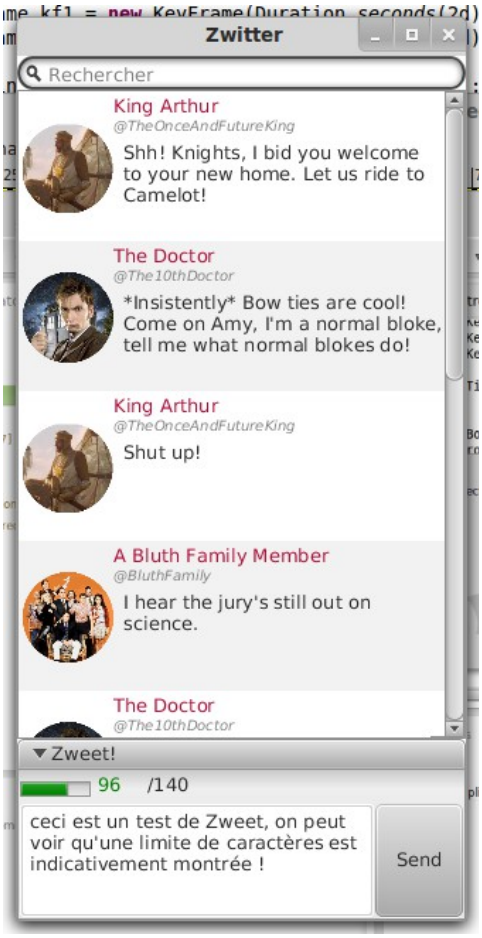
Avant-propos

Nous allons créer une application nommée Zwitter qui a pour but d'afficher des « zweets » un peu à l'image d'un client twitter avec des tweets.

Le but est évidemment de se familiariser avec les API JavaFX mais aussi de comprendre la démarche de création d'écran riche.

L'application Zwitter reçoit périodiquement des zweets via une API fournie qui ne se sera pas détaillée ; elle permettra aussi de saisir des messages dans une timeline.

Ces travaux pratiques ne sont pas spécifiques à un IDE et peuvent donc être réalisés sous Eclipse ou Intellij selon l'envie.



TP 0 : Hello World !

1. Objectifs

- Utiliser SceneBuilder avec e(fx)clipse / Intellij
- Réaliser un simple hello world

2. Installer et configurer Scene Builder

L'outil est disponible sur le site d'Oracle :
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Télécharger et installez la version qui correspond à votre système.

Afin de pouvoir appeler SceneBuilder depuis Eclipse facilement, il suffit de le rajouter dans la configuration de e(fx)clipse :

```
Window > Preferences... > JavaFX > Browse... > indiquer
l'emplacement de SceneBuilder
```

Une fois cette manipulation effectuée, dans la vue Package Hierarchy on pourra faire un clic-droit sur un fichier .fxml et "Open with SceneBuilder".

3. Hello World

1. Utiliser le wizard d'Eclipse `New Project > JavaFX > New JavaFX Project...`

2. Déclarer vouloir utiliser un FXML (exemple : `MainController`, `mainScreen.fxml`) dans le 3ème et dernier écran du wizard, puis ouvrir mainScreen.fxml avec SceneBuilder.

3. Y construire la fenêtre principale avec un simple bouton, dont le clic doit appeler une méthode "showPopup". Coder le contrôleur et sa méthode publique `showPopup()`.

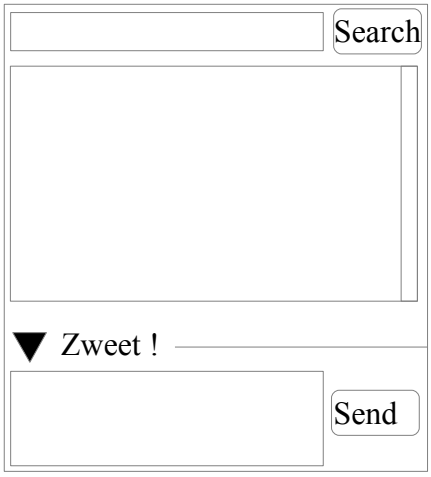
TP 1 : Squelette de l'application

1. Objectifs

- Editer une vue en FXML
- Se familiariser avec les composants JavaFX
- Lier FXML et contrôleur

2. La fenêtre principale

Layout désiré



1. Editer un fichier fxml nommé MainScreen.fxml soit avec SceneBuilder, soit via un éditeur XML (celui de l'IDE par exemple). Ce FXML est basé sur le composant racine BorderPane.

2. Nous souhaitons obtenir une fenêtre découpée en 3 parties : une zone de recherche en haut, une zone d'affichage des Zweets au centre (le gros de l'écran) et une zone d'écriture de nouveaux Zweets en bas. Nous définirons donc 3 conteneurs différents aux parties « bottom » « center » et « top » du BorderPane.

3. La partie haute de l'application regroupe un champ de saisie ainsi qu'un bouton de recherche alignés horizontalement. On utilisera donc un conteneur HBox pour arranger les éléments de cette partie. Le champ de saisie doit pouvoir occuper tout l'espace disponible horizontalement, le bouton utilisera sa taille par défaut.

4. La zone centrale est l'emplacement d'un composant ListView (tableau simple en JavaFX). Pour permettre plus de liberté en cas d'évolution, ce composant ListView sera contenu dans un composant AnchorPane. Afin de pouvoir utiliser par la suite le composant ListView, on lui affecte un fx:id à la valeur « timeline ».

5. Pour la partie inférieure avec saisie de zweet, on utilisera un HBox. Dedans, on placera un champs de saisie multi-ligne qui prendra le plus de place posssible ainsi qu'un bouton d'envoi.

6. Afin de rendre la zone masquable, on décide d'utiliser un TitledPane dans la partie inférieure de l'écran. Modifier le FXML afin de l'afficher correctement.

3. Utiliser ZwitterAPI, lier le FXML et le contrôleur

1. Injecter le composant ListView dans le contrôleur Java soit par son identifiant public, soit via une annotation @FXML.

2. Implémenter l'interface Initializable de JavaFX dans le contrôleur Java, la méthode initialize(...) sera appelée après l'initialisation de tous les composants JavaFX définis en FXML (ils ne seront pas nuls).

3. Définir une collection bindable pour JavaFX dans le contrôleur :

   ```
   ObservableList<Zweet> list =
           FXCollections.observableList(new LinkedList<Zweet>());
   ```

4. Pour utiliser un « Zwitter » on utilise une API fluent avec la classe ZwitterBuilder.

   ```
   Zwitter zwitter = ZwitterBuilder.create().withObservableList(list).build();
   ```

   On peut préciser la liste qui sera remplie des Zweets via la méthode withObservableList(...), on peut aussi préciser un nombre maximum d'éléments dans la liste.

5. Affecter la collection au composant JavaFX via la méthode setItems :

   ```
   listView.setItems(list);
   ```

6. Démarrer Zwitter avec :

   ```
   zwitter.start();
   ```

7. Tester.

8. Stopper zwitter lorsque l'application se termine avec :

   ```
   zwitter.stop() ;
   ```

TP 2 : CellFactory et Databinding

1. Objectifs

- Comprendre et utiliser le système d'item-renderer / Cell Factory
- Comprendre et utiliser un databinding simple
- Utiliser une binding expression simple

2. Utiliser CellFactory et personnaliser les lignes d'un tableau

1. Créer une nouvelle classe nommée ZweetCell qui implémente l'interface Callback<ListView<Zweet>, ListCell<Zweet>>.

2. La méthode call(...) est appelée par JavaFX pour définir le rendu de chaque ligne de tableau lors d'un changement de donnée. Elle requiert de définir l'instance de type ListCell<Zweet> qui sera utilisée pour le rendu. Dans un premier temps retourner une instance de type TextFieldListCell<Zweet> :

   ```
   final ListCell<Zweet> cell = new TextFieldListCell<Zweet>();
   return cell;
   ```

3. Affecter cette classe comme CellFactory à notre composant ListView :

   ```
   timeline.setCellFactory(new ZweetCell());
   ```

4. Tester l'affichage.

5. Dans la classe ZweetCell, plutôt que de retourner un TextFieldListCell<Zweet>, on décide de retourner une instance d'une nouvelle classe qui étend ListCell<Zweet>. Celle-ci devra surcharger la méthode updateItem(...).

   ```
    @Override
   public void updateItem(Zweet item, boolean empty) {
           super.updateItem(item, empty);
           Text text = new Text();
           text.setText("un texte !");
           setGraphic(text);
   }
   ```

6. Dans cette méthode le paramètre empty précise si des données sont présentes et le paramètre Zweet possède la valeur « modèle » de la ligne (null si empty est vrai). Si empty est faux : le CellFactory a pour responsabilité de dessiner le rendu des données.

   Ajouter une instance de type Text à la classe et instancier ce composant si des données sont présentes et que sa valeur est nulle. Affecter la valeur du texte à afficher à zweet.toString().

7. Tester le rendu.

8. Afin d'optimiser l'implémentation de la CellFactory, il n'est pas nécessaire d'instancier à chaque Zweet un nouveau Text si celui-ci a déjà été créé. Modifier l'implémentation de ZweetCell pour qu'il n'utilise qu'un seul composant Text qui sera défini comme variable d'instance.

9. Tester le rendu.

3. Databinding

1. Il est actuellement possible de redimensionner la taille de l'application JavaFX, la valeur maximale du composant Text est en réalité dynamique ! Définir un databinding entre le composant ListView et le composant Text afin qu'ils partagent les mêmes contraintes de largeur (propriété wrappingWidth pour Text et widthProperty pour ListView).

2. Tester le rendu avec plusieurs redimensionnements de fenêtre.

3. On souhaite toujours avoir un écart entre le composant Text et la scrollbar verticale à droite, définir une binding expression sur wrappingWidth pour que cet écart soit toujours de 20 pixels.

4. Tester le rendu avec plusieurs redimensionnements de fenêtre.

TP 3 : Composant en Java

1. Objectifs

- Définir un composant en Java
- Définir un composant en FXML avec racine personnalisée via fx:root
- Utiliser un composant personnalisé dans une CellFactory
- Utiliser le layout GridPane

2. CellFactory et composant personnalisé en Java

1. Créer un nouveau composant Java nommé ZweetPane dans le package `com.zenika.fx.zwitter.zweetui`.

2. Définir le rendu de ce composant selon l'image ci-dessous à l'aide d'un GridPane en Java :

3. Dans ce composant Java, définir une méthode setZweet(Zweet zweet) qui affiche le contenu du Zweet dans le composant.

4. Utiliser ce composant à la place du composant Text défini dans la CellFactory du TP2.

5. Tester le rendu.

3. Utilisation de fx:root

1. On désire cette fois-ci implémenter ce composant à l'aide de FXML, ajouter un nouveau fichier FXML nommé zweetPane.fxml au projet et y définir le rendu.

2. Afin de pouvoir utiliser ce fichier FXML dans un CellFactory, il est nécessaire d'utiliser un design pattern de type « MVVM » : modifier la classe Java ZweetPane et charger le FXML défini précedemment à l'aide de FXMLLoader dans le constructeur de ZweetPane :

```
FXMLLoader fxmlLoader = new
        FXMLLoader(getClass().getResource("zweetPane.fxml"));
fxmlLoader.setRoot(this);
fxmlLoader.setController(this);
try {
        fxmlLoader.load();
} catch (IOException e) {
        e.printStackTrace();
}
```

3. Dans le fichier FXML, utiliser la balise <fx:root> comme racine et spécifier l'attribut « type » à la valeur javafx.scene.layout.GridPane.

4. Injecter les composants FXML dans ZweetPane avec l'annotation @FXML. Retirer le code de construction de vue de ZweetPane réalisée dans la partie 2 de ce TP.

5. Modifier le constructeur de ZweetPane afin qu'il prenne un Zweet comme paramètre qui sera stocké en tant que variable d'instance.

6. Implémenter l'interface Initializable et appeler setZweet(zweet) dedans où zweet est la valeur du Zweet utilisé pour la construction de ZweetPane.

7. Tester le rendu.

TP 4 : CSS, Effets et Transitions

1. Objectifs

- Utiliser des feuilles de style CSS avec JavaFX
- Créer une animation personnalisée
- Intégrer une animation avec CellFactory

2. CSS

1. Afin de supprimer les possibles scrollbars horizontales de notre composant ListView, définir dans le fichier CSS de l'application un -fx-padding à la valeur 0 pour les scrollbars horizontales de ListView.

   Les propriétés concernées sont :

   .list-view .scroll-bar:horizontal .increment-arrow,
   .list-view .scroll-bar:horizontal .decrement-arrow,
   .list-view .scroll-bar:horizontal .increment-button,
   .list-view .scroll-bar:horizontal .decrement-button

2. Tester le rendu.

3. On désire désormais que ces propriétés ne concernent que notre ListView. Après tout, il est possible que quelqu'un d'autre veuille utiliser un ListView avec des scrollbars dans notre application. Définir une classe CSS et l'affecter correctement dans le fichier FXML.

4. Tester le rendu.

3. Animations via transitions

1. Définir une méthode statique publique nommée createTransition(Node node) retournant un objet Transition pour l'instant nul.

2. Dans cette méthode, instancier une FadeTransition durant 1 seconde allant d'invisible à visible s'appliquant au node donné en paramètre et la retourner comme résultat.

3. Utiliser cette transition sur un élément de l'application en instanciant une transition via le méthode statique précédente et en executant la méthode play() sur l'objet transition.

4. On désire désormais utiliser cet effet sur les ZweetPane apparaîssant dans notre ListView. Instancier dans chaque ZweetPane, un objet transition qui s'exécutera sur ce composant via la méthode play() dès que le contenu de type Zweet change.

5. Tester l'affichage.

6. Il est possible d'utiliser et de combiner plusieurs effets, utiliser une ScaleTransition allant de l'échelle 0 à l'échelle 1 durant 1 seconde à la place de la FadeTransition.

7. Tester l'agrandissement.

8. A l'aide de la classe ParallelTransition, appliquer les deux effets simultanément.

9. Tester.

10. A l'aide de SequentialTransition, appliquer les deux effets consécutivement.

11. Tester.

12. Cet effet ne doit s'appliquer que sur les nouveaux Zweets, modifier ZweetPane pour cela.

TP 5 : Associer un comportement à l'IHM

1. Objectifs

- Ajouter des listeners de différentes manières
- Utiliser des propriétés avancées CSS

2. Produire des Zweets

1. Dans la partie inférieure de l'application, lier le clic du bouton à une fonction du contrôleur, cette méthode va nous permettre d'émettre des Zweets.

2. Dans cette méthode, forger un Zweet et utiliser l'API Zwitter afin de publier un Zweet via la fonction publish(Zweet zweet). Afin de forger un Zweet correct, il faudra valuer un objet ZweetUser avec un username et displayname au choix et en utilisant comme image de profil "znk.png".

3. Tester l'emission d'un Zweet.

4. Afin de différencier l'animation d'un Zweet « externe » d'un Zweet produit, modifier l'implémentation de ZweetPane afin que l'animation dure moins de temps pour un Zweet produit.

5. Tester.

3. Composant barre de recherche

1. Créer un nouveau composant nommé SearchBox étendant la classe Region.

2. Ce composant possède un champs texte et un bouton qui seront des variables d'instance. Ajouter ces composants au rendu dans le constructeur de SearchBox.

3. Intégrer ce composant dans le fichier FXML de l'application pour remplacer la partie haute dédiée à la recherche.

4. Tester le rendu.

5. Lors d'un clic sur le bouton, le texte doit être effacé. Ajouter un listener en Java dans SearchBox afin d'effectuer cette action.

6. Le bouton ne doit être visible que si le texte saisi n'est pas vide. Ajouter un binding afin de masquer le bouton quand le texte est vide.

7. Tester le composant SearchBox avec quelques saisies de texte.

8. Afin de bénéficier d'un redimensionnement automatique, ajouter les bindings adéquats dans l'implémentation de SearchBox entre les diverses largeurs / hauteurs possibles.

9. Tester plusieurs redimensionnements et vérifier que le texte de saisie se redimensionne correctement.

10. Pour enrichir le rendu de cette barre de recherche, utiliser les images fournies afin d'ajouter une loupe et modifier le rendu du bouton. Pour cela, on définira des classes CSS. On utilisera notamment les propriétés suivantes pour le rendu de la saisie :

    -fx-background-image: url('images/middle.png'), url('images/left.png'), url('images/right.png') ;
    -fx-background-position: ...
    -fx-background-repeat: ...

11. Tester le rendu.

TP 6 : Binding Avancé

1. Objectifs

- Définir un Binding personnalisé
- Utiliser un binding à haut niveau
- Utiliser un binding à bas niveau en l'observant

2. Ecouter les événements de saisie

1. Ajouter dans la partie inférieure de l'application un champ texte. Celui-ci permettra de suivre le nombre de caractères saisi dans le champ de Zweet en affichant par exemple « 3 / 140 ».

2. Définir un binding dans le contrôleur de type IntegerBinding, ce binding permettra de suivre la saisie d'un Zweet (nombre de caractères) jusqu'à un maximum de 140.

3. Afin de pouvoir lier cet IntegerBinding au textProperty il faudra surcharger la méthode computeValue() de IntegerBinding afin que la valeur de l'IntegerBinding corresponde à la longueur de texte saisi dans textProperty.

4. Binder l'IntegerBinding personnalisé avec le TextArea (textProperty).

5. Tester à l'aide de la console / ou au mode debug que la valeur du binding est correcte.

6. Binder désormais l'IntegerBinding au texte dédié afin de montrer le nombre de caractères saisis.

7. Tester le rendu.

8. On désire ajouter une barre de progression afin de montrer l'évolution de la saisie (de 0 à 100%). Ajouter un composant ProgressBar.

9. Binder ce composant ProgressBar avec IntegerBinding sachant que 100 % = 140 caractères.

10. Tester le rendu.

11. On désire changer l'apparence du texte saisi et du texte suivant le nombre de caractères si la saisie dépasse 140 caractères. Pour cela, affecter une classe CSS à ces composants.

12. Écouter l'évolution de la saisie et changer la classe CSS de ces composants dès que la saisie dépasse 140 caractères (ou passe en dessous, il faut prévoir les deux cas).

13. Tester le rendu.

TP 7 : API Task

1. Objectifs

- Découvrir l'API Worker et plus particulièrement l'implémentation Task
- Déclencher une Task / Annuler une Task
- Réagir à la complétion ou l'échec d'une Task

2. Implémentation d'une Task

1. Créer une nouvelle classe nommée SearchTask qui étend Task<Set<Zweet>>.

2. Cette Task a besoin de deux paramètres pour être construite, une collection de Zweet nommée zweets et une String nommée keyword. Ces deux éléments permettront de rechercher un texte dans une collection de Zweets et de retourner le résultat sous forme de Set.

3. Implémenter la méthode call() qui renvoie le Set des Zweets contenus dans la collection qui contiennent le String keyword.

4. Afin de simuler une recherche lente, ajouter un Thread.sleep(1000) dans la méthode call().

5. Ajouter un listener dans le champs de saisie de SearchBox afin d'exécuter une Task SearchTask quand il y a de nouveaux caractères saisis.

6. Définir un handler onSucceededHandler qui affecte le Set des zweets filtrés comme source à notre composant ListView.

7. Tester le filtrage.

8. Afin de revenir à la vue de tous les Zweets, réaffecter la collection de Zweets originale quand il n'y a plus de caractères saisis dans SearchBox.

9. Tester.

3. Optimisation

1. Comme chaque caractère saisi va produire une recherche, on cherche à optimiser ce traitement : lancer la recherche seulement si au moins 3 caractères sont saisis.

2. Tester.

3. Comme chaque caractère au-delà de trois va lancer une recherche, seule la dernière recherche lancée n'a d'intérêt. Il faut donc annuler via cancel() la Task en cours si elle est toujours en traitement, avant de lancer une nouvelle recherche.

4. Tester.

5. Vérifier les différents cas d'utilisation et traiter les derniers cas particuliers (recherche annulée, aucun caractère saisi).

6. Tester.

TP 8 : API Service

1. Objectifs

- Découvrir l'API Service et les facilitations qu'il apporte
- Convertir / réutiliser une Task dans un Service
- Utiliser les bindings pour paramétrer chaque exécution du Service

2. Conversion vers l'API Service

Plutôt qu'explicitement créer un thread pour l'exécution de la SearchTask, devoir garder trace des tâches en cours pour pouvoir les annuler, etc... il est plus intéressant de s'appuyer sur un Service.

Nous allons donc définir un SearchService, qui aura pour rôle d'instancier des SearchTask, dans le package `com.zenika.fx.zwitter.search` lui aussi.

3. Paramétrage du Service

La SearchTask était créée explicitement par nous, à un endroit où nous avions accès au champs texte et à la timeline. Nous devons maintenant laisser soin au service de passer ces paramètres au constructeur de la Task.

Nous allons avoir besoin de deux propriétés pour le mot-clé et la liste de Zweets, que nous capturerons au moment de la création de la tâche en elle-même, et auxquelles nous affecterons la valeur correcte (éventuellement par binding) à l'appel du service.