TP 2 – Architecture d'applications – Architecture MVC et pattern DAO dans une application

Dans ce TP, nous allons commencer à aborder la notion d'architecture d'applications par la création d'une application à la fois monolithique et standalone (un exécutable, par opposition à une application web).

Pour cela, nous allons créer une petite application de calcul de périmètre d'un rectangle en JavaFX.

L'objectif de ce TP n'est pas d'apprendre JavaFX (bien que cela ne fasse pas de mal!) mais plutôt d'expérimenter la mise en œuvre classique d'une application avec interface graphique.

Etape 1 – Mise en place de l'environnement

JavaFX est une bibliothèque Java permettant de réaliser des interfaces graphiques. D'autres bibliothèques existent (notamment swing et awt, plus anciennes). JavaFX fait partie intégrante de Java jusqu'à sa version JDK 10. A partir du JDK11, il devient nécessaire de télécharger le SDK open-source de JavaFX en plus du JDK.

Nous utiliserons IntelliJ pour développer en JavaFX, pour la simple et bonne raison que cet éditeur propose une intégration relativement directe et simple (selon votre version du JDK) de JavaFX, et notamment de l'éditeur graphique d'interfaces que nous utiliserons.

1- Commencez donc par télécharger et installer IntelliJ (version community) si vous ne l'avez pas déjà : https://www.jetbrains.com/fr-fr/idea/download/#section=windows

2- Vérifiez ensuite que vous avez bien le JDK d'intallé, ainsi que votre version du JDK. Pour cela, vous pouvez ouvrir le terminal et taper

```
javac -version
```

- Voici un exemple de ce que l'on obtient.

```
C:\Users\Proprietaire>java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)

C:\Users\Proprietaire>javac -version
javac 1.8.0_251
```

Si vous n'avez pas de JDK installé, allez installer la version 8 : cela évitera des complications

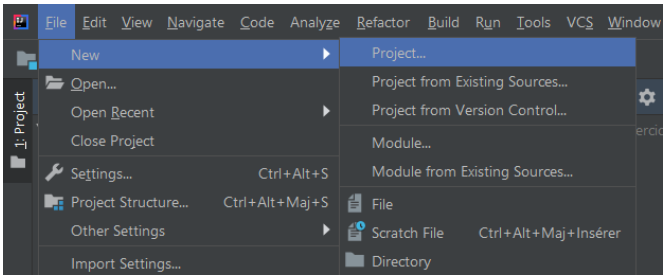
https://downzen.com/fr/windows/java-development-kit-jdk/download/8-update-251/

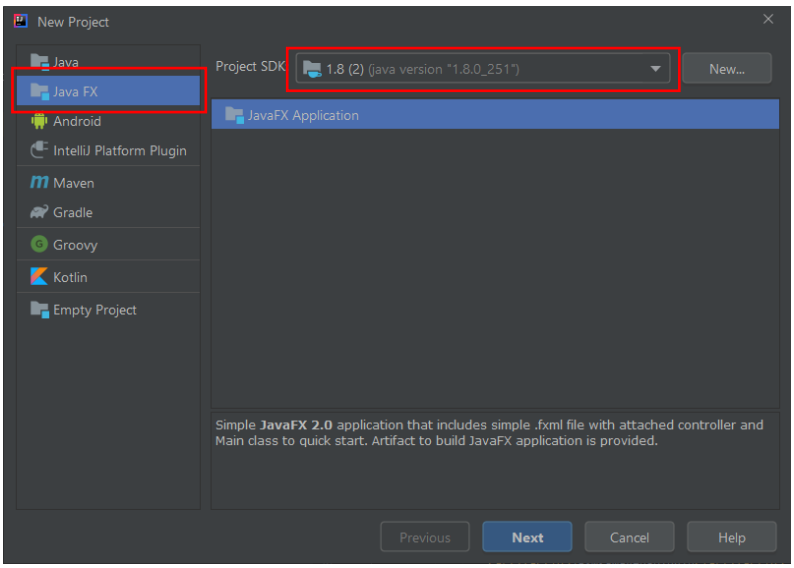
3- Si vous avez une version du JDK supérieure à 10, installez JavaFX en suivant les instructions ici:

https://www.jetbrains.com/help/idea/javafx.html

Etape 2 – Créer un nouveau projet

Encore une fois, référez vous à https://www.jetbrains.com/help/idea/javafx.html pour les manipulations à effectuer pour créer un nouveau projet. Si votre JDK est le JDK10 ou inférieur, vous n'avez rien à faire: dans IntelliJ, il suffit de créer un nouveau projet (File > New Project) puis de choisir le format "JavaFX" à gauche de la fenêtre. Vérifiez bien que votre SDK de projet correspond à la version que vous souhaitez.





Donnez ensuite un nom et un emplacement à votre projet et validez.

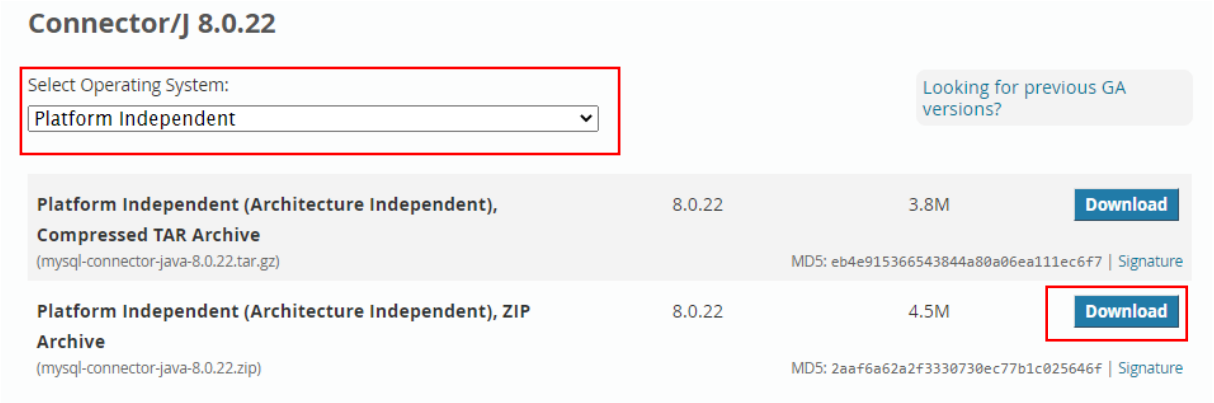
Installez ensuite le connecteur mysql-connector comme décrit dans les diapositives préalables à ce TP.

Etape 3 – Ajouter le connecteur MySQL à votre projet

Commencez par télécharger le connecteur mySQL à l'adresse suivante

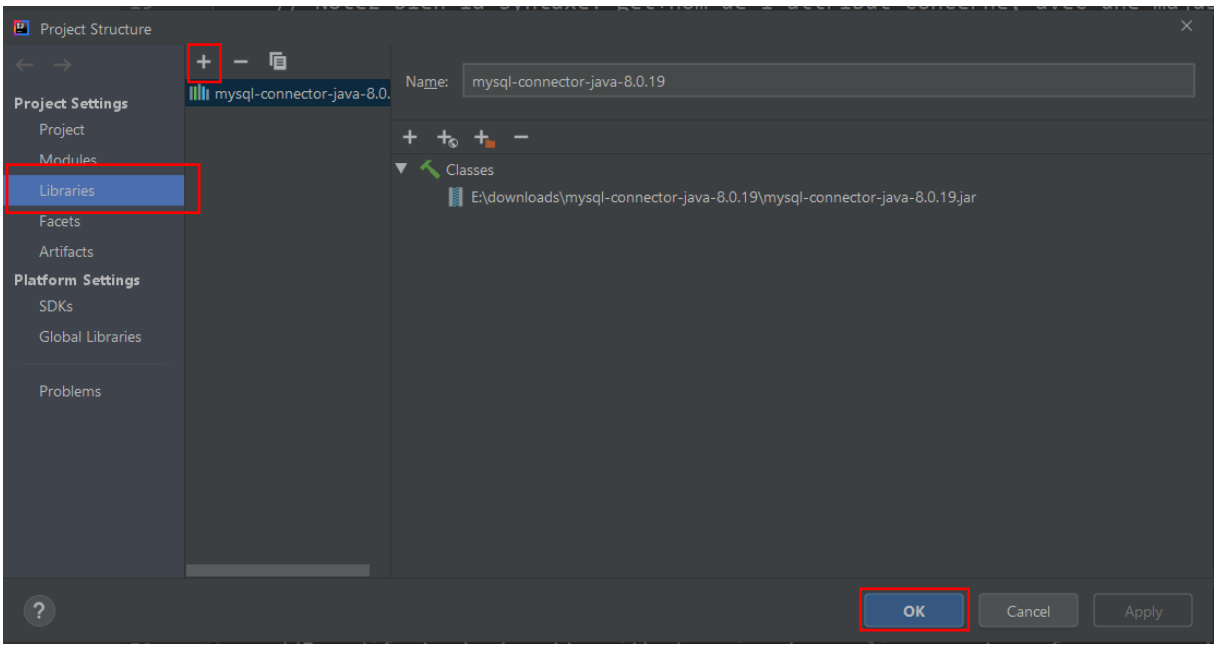
https://dev.mysql.com/downloads/connector/j/

Sélectionnez "Platform Independent" et téléchargez l'archive ZIP:



Dézippez le fichier ainsi obtenu.

Dans votre projet IntelliJ, alez dans **File > Project** Structure et cliquez sur **Librairies**.



Cliquez sur le + en haut à gauche et ajouter le fichier mysql-connector-java….jar.

Cliquez sur OK.

Exercice – Créer une application structurée autour d'une GUI et d'une base de données

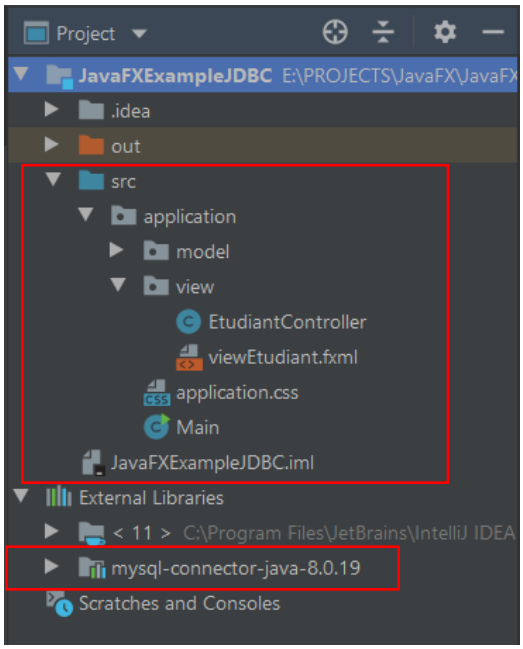
Dans cet exercice, nous allons créer une application très simple de listing d'étudiants par filière, et permettant d'afficher un étudiant sur une interface graphique. Ce TP nous permettra de mettre en œuvre une architecture classique : le pattern MVC associé au pattern DAO pour la partie modèle.

Création des packages

Afin de mieux structurer notre application, refactorez le package "sample" en un package "application". Puis, au sein de ce package application, créez deux packages "view" et "model".

Faites un refactor ensuite de la vue fxml en viewEtudiant.fxml ; et du Controller en EtudiantController.fxml. Placez ces deux fichiers dans le package view.

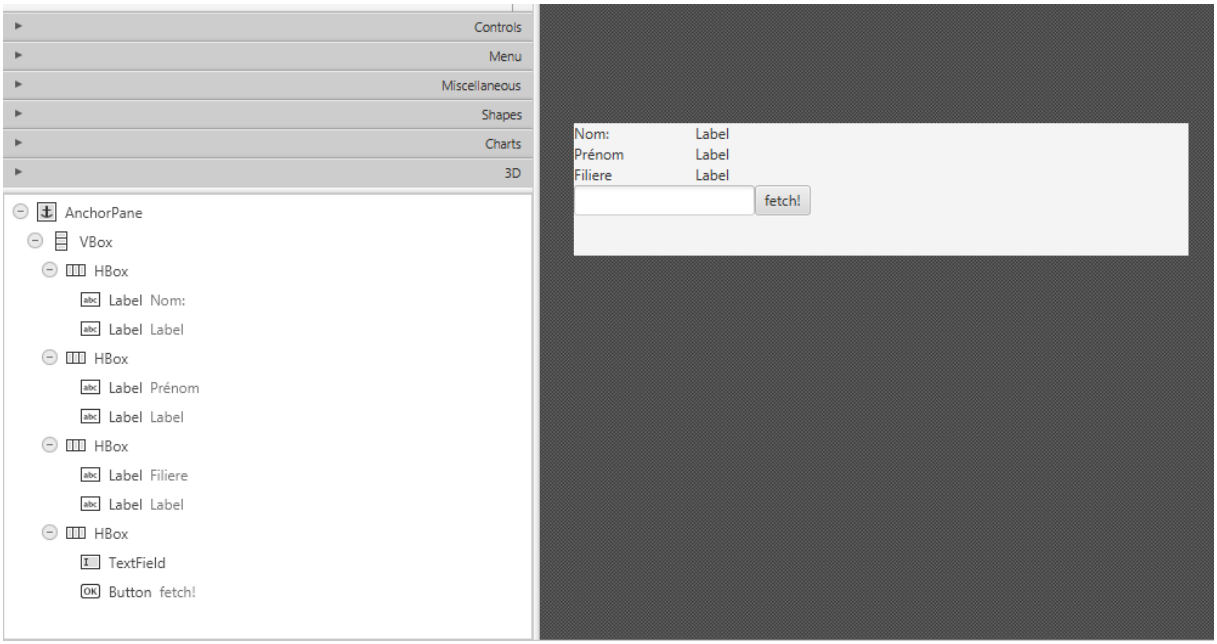
Vous devez alors avoir l'arborescence suivante :



Notez que vous devez voir apparaitre le connecteur mysql dans vos librairies externes.

Création de la vue et du controller

Grâce à l'éditeur SceneBuilder, créez la vue suivante :



Le TextField permettra à l'utilisateur de saisir un id d'étudiant pour voir apparaître son nom, prénom et filière dans les labels correspondants.

Puis, dans votre fichier EtudiantController.java, vous allez ajouter une variable correspondant au premier label de l'interface, que vous aurez besoin de manipuler pour afficher le nom :

```
@FXML
private Label label1;
```

Faites ensuite de même pour les labels du prénom (label2) et de la filière (label3), et le TextField (tf). N'oubliez pas de préfixer vos variables avec l'annotation @FXML.

Création du modèle

Nous allons ici architecturer notre modèle selon le pattern DAO.

Dans le package model, créez tout d'abord un package beans et un package dao. Dans le package beans, nous rangerons toute nos classes "beans", c'est-à-dire correspondant à une table de notre base de données ; dans notre cas, nous ne créerons qu'un bean "Etudiant". Dans le package dao, nous rangerons toutes les classes permettant un accès à la base de données.

Dans le package bean, créez une classe Etudiant. Cette classe contiendra:

- Un attribut id (int)
- Un attribut nom (String)
- Un attribut prenom (String)
- Un attribut filiere (String)

Créez les get/Set pour ces différents attributs, ainsi qu'une méthode toString() retournant une chaîne de caractères décrivant un étudiant (son id, son nom/prénom, sa filière).

Dans le package dao, créez une classe MySqlConnection et copiez-collez le code suivant:

```
package application.model.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;


// IL EST POSSIBLE DE COPIER COLLER LE CODE DE CETTE CLASSE DANS VOTRE PROJET
// AINSI QUE LA CLASSE TEMPLATE DAO<T>
// VEILLEZ SIMPLEMENT A MODIFIER LES PARAMETRES DE CONNEXION A LA BDD


//Classe unique permettant d'assurer la connexion à la base de données.
// Pas grand chose à changer ici, juste les paramètres de connexion.
// Le pattern singleton représenté ici permet de ne pas avoir à constamment
// créer des connexions vers la BDD.
// vous pouvez vous passer de cette classe si vous vous connectez à chaque fois
// à la BDD dans les différents DAO.
public class MySqlConnection {

   /** * paramètres de connexion:  URL de connection et login, pass pour la BDD */
   /* A réutiliser tel quel, juste ces 3 paramètres à changer */
   // il faut garder les paramètres ajoutés en fin d'URL; gardez donc bien ce bout
lorsque vous modifier le host, port et nom de Bdd:
   //
jdbc:mysql://ipdelhote:numerodeport/nomdelabase?useUnicode=true&useJDBCCompliantTi
mezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
   private static String url =
"jdbc:mysql://localhost:3306/testjdbc?useUnicode=true&useJDBCCompliantTimezoneShif
```

```
t=true&useLegacyDatetimeCode=false&serverTimezone=UTC";
   private static String user = "root";
   private static String passwd = "";

   /** * Objet Connection */
   private static Connection connect;

   /** * Méthode qui va nous retourner notre instance * et la créer si elle
n'existe pas... * @return */
   public static Connection getInstance(){
       if(connect == null){
      try {
          connect = DriverManager.getConnection(url, user, passwd);
      }
      catch (SQLException e) {
         e.printStackTrace();
      }
       }
       return connect;
   }


}
```

Pensez à modifier les variables statiques url, user et passwd pour vous connecter à votre base de données.

Dans le package dao, créez une classe DAO et copiez-collez le code suivant:

```
package application.model.dao;

import java.sql.Connection;

// IL EST POSSIBLE DE COPIER COLLER LE CODE DE CETTE CLASSE DANS VOTRE PROJET
// AINSI QUE LA CLASSE MySqlConnection


//Ma classe mère, abstraite, et template pour le DAO. Rien à toucher ici.
public abstract class DAO<T> {

   //Je récupère une instance de connexion grace à la méthode statique
getInstance()
   // de MySqlConnection (cf classe MySqlConnection)
    public Connection connect = MySqlConnection.getInstance();


    //Puis les méthodes "standard" pour le CRUD sur une entité.
    /** * Permet de récupérer un objet via son ID * @param id * @return */
    public abstract T find(long id);

    /** * Permet de créer une entrée dans la base de données * par rapport à un
objet*/
    public abstract T create(T obj);

    /** * Permet de mettre à jour les données d'une entrée dans la base  */
    public abstract T update(T obj);
```

```
    /** * Permet la suppression d'une entrée de la base * @param obj */
    public abstract void delete(T obj);

}
```

Cette classe abstraite permet de factoriser la connexion à la base de données entre les différents DAO des beans de notre application, et de spécifier des méthodes de base correspondant à un CRUD.

Dans notre cas, nous n'avons qu'un seul bean Etudiant et donc qu'un seul DAO EtudiantDAO, ce qui fait que cette classe abstraite n'est pas très intéressante… mais c'est une bonne pratique.

Enfin, toujours dans le package dao, créez une classe EtudiantDAO, copiez-collez le code suivant et remplissez les passages notés //TODO :

```
package application.model.dao;

import application.model.beans.Etudiant;

import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

//DAO (Data Access Object) pour le bean Etudiant.
/*
 * On va ranger dans cette classe tout le code permettant de faire le lient avec
la table
 * Etudiant dans notre base de données. Pour cela, on va extensivement utiliser
 * notre bean Etudiant.
 *
 * On va retrouver dans le DAO les méthodes abstraites du template DAO<T>:
 * - find - encapsule des SELECT
 * - create - Encapsule un INSERT
 * - update - encapsule un UPDATE
 * - delete - encapsule un requete DELETE
 *
 * On trouvera très souvent plusieurs méthodes find() qui permettront de faire des
requetes diverses
 * (Select...) sur la BDD.
 *
 */
// Herite de la classe abstraite DAO<Etudiant>
public class EtudiantDAO extends DAO<Etudiant> {

   //Premiere méthode find. Renvoie un Etudiant, connaissant son identifiant.
   @Override
   public Etudiant find(long id) {
      //Pour pouvoir créer un objet etudiant à partir d'un enregistrement en base

      //Je commence par créer un objet Etudiant (vide):
      Etudiant etud = new Etudiant();

      // Puis je vais faire un select sur la bdd pour récupérer l'enregistrement
d'identifiant
```

```
      // correspondant au paramètre id
      try {
         Statement stmt = connect.createStatement();
         ResultSet rs = stmt.executeQuery( //TODO : écrire la requête SQL
                              );   // c'est l'id donné en paramètre de la méthode
         while (rs.next()) {               // Tant que j'ai des résultats (mais
normalement y'en a qu'un)
            //Je récupère les données de ma base (en fait de mon ResultSet)
            int bd_id = rs.getInt("id");
            //TODO avec le nom, prenom, filière
            //Et je remplis mon objet etud avec les données récupérée depuis la
BDD
            //TODO
            //mon objet étudiant contient bien les informations correspondant à
l'enregistrement recherché.
         }

         //Je retourne mon objet correctement rempli
         return etud;
      }
      catch (Exception e) {
         System.out.println("EtudiantDAO: find() failed:
"+e.getLocalizedMessage());
      }

      return null;
   }


   //Méthode de création d'un étudiant, cad insertion d'un nouvel étudiant dans la
base.
   // Je prends en parmaètre l'objet Etudiant qui contient les informations à
insérer.
   @Override
   public Etudiant create(Etudiant obj) {
      // TODO Auto-generated method stub
      return null;
   }

   // TODO: reprendre le créate pour cette fois mettre à jour un étudiant et le
retourner.
   @Override
   public Etudiant update(Etudiant obj) {
      // TODO Auto-generated method stub
      return null;
   }

   //TODO: reprendre le create pour effacer un étudiant (on ne le retourne pas du
coup)
   @Override
   public void delete(Etudiant obj) {
      // TODO Auto-generated method stub

   }

}
```

Reprise du contrôleur et de la vue pour utiliser le modèle

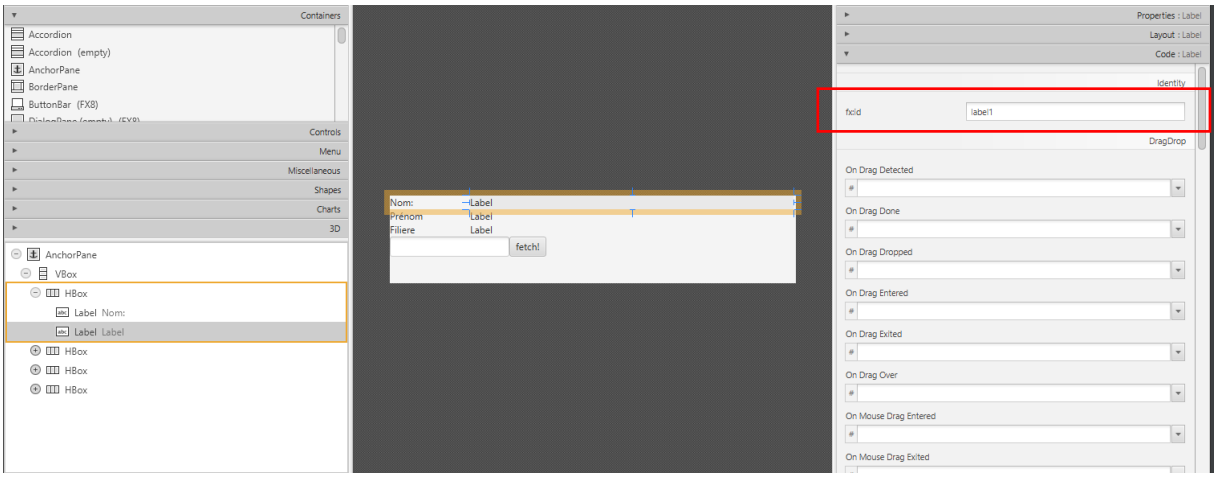
Maintenant que votre modèle est réalisé, nous allons pouvoir le lier avec la vue grâce au contrôleur.

Créez tout d'abord une méthode fillView(Etudiant e) dans votre EtudiantController, qui permettra de mettre à jour vos labels avec les informations d'un étudiant (méthode setText(String txt) de la classe Label).

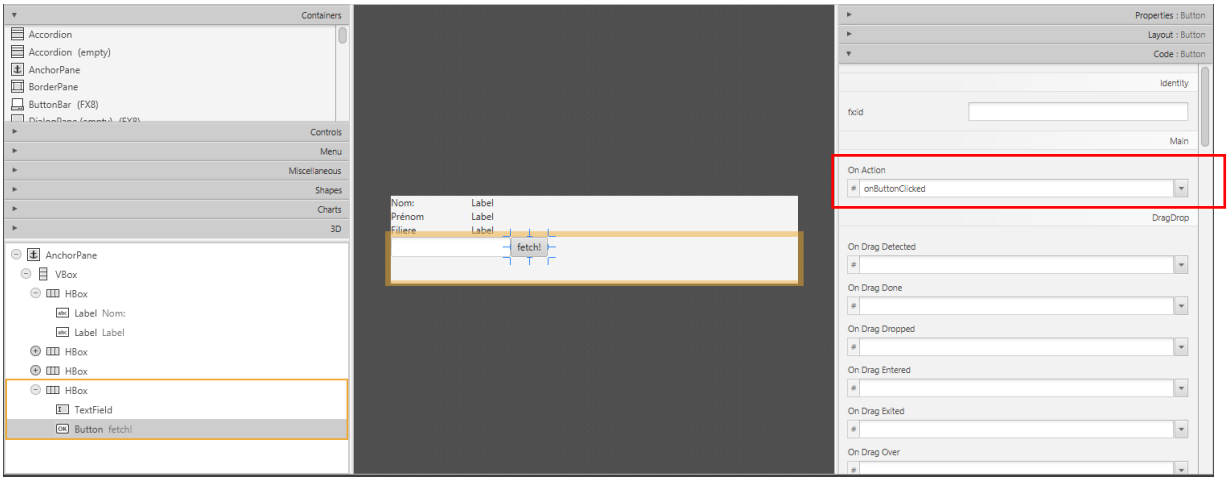
Créez ensuite une méthode onButtonClicked(), , profixée par l'annotation @FXML, qui va

1- Récupérer le texte de votre variable tf (méthode getText() de la classe TextField) et le transformer en int
2- Instancier un nouvel EtudiantDAO ;
3- Utiliser la méthode find() sur cette instance d'EtudiantDAO et récupérer ainsi un objet étudiant
4- Mettre à jour la vue avec cet étudiant grâce à la méthode fillView.

Enfin, reprenez votre vue viewEtudiant.fxml. Affectez à vos labels et au TextField les bons fx:id :



Et affectez à l'évènement OnAction de votre bouton la méthode onButtonClicked():



Et voilà! Vous n'avez plus qu'à créer votre base de donnée, une table étudiant, remplir quelques lignes… et le tour et joué!

```
DROP TABLE IF EXISTS `etudiant`;
CREATE TABLE IF NOT EXISTS `etudiant` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `Nom` varchar(255) NOT NULL,
  `Prenom` varchar(255) NOT NULL,
  `fil` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Pour aller un (tout petit peu) plus loin…

Modifiez votre programme et inspirez vous des méthodes données pour permettre également de créer une nouvel étudiant en base de données.

Il faudra pour cela implémenter la méthode create dans votre EtudiantDAO. Attention, la méthode java pour exécuter un INSERT est execute() (alors que nous avions utilisé un executeQuery() pour un SELECT).