**Cours 2 octobre 2017**

**JAVA FX**

Professeur : Thibaud de Mijolla

Java FX

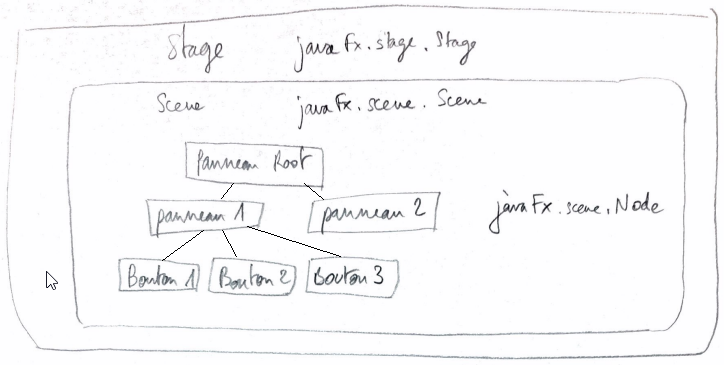
\* Java Special Effect

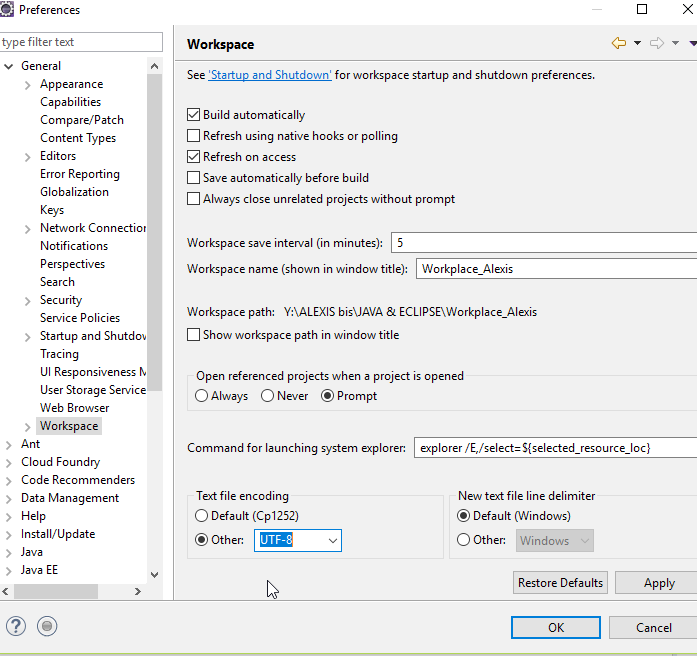
\* Librairie officielle de conposants graphiques Java  
- client lourd  
- client léger (web)

\* remplace :  
- AWT : Abstract Windows Toolkit (dès 1995, Java 1.0)  
- Swing (1998 → Java 1.2)

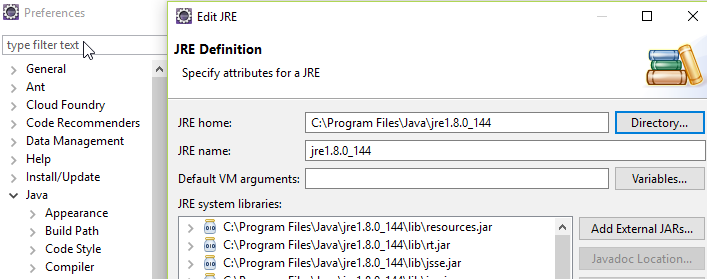
\* Historiquement :  
2008-2011 : JavaFXScript  
2011-2014 : JavaFX2  
2014 : JavaFX8

Java Stage OCR ([clic](https://openclassrooms.com/courses/les-applications-web-avec-javafx/la-scene-graphique-de-javafx))





Vérifier aussi qu’on est toujours en UTF8 et dans la bonne bibliothèque (1.8)



**Le pading et le marging**

Le pading est entre le conteneur et l’objet à l’intérieur,  
Le marging est entre les objets !

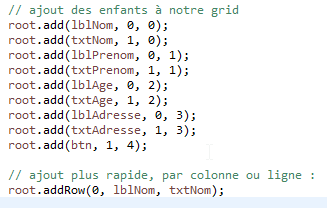
Le pading est la marge par rapport aux éléments que je contiens.



Sur le PaneFlow :



Quant au GridPane :



**ATTENTION** en ajoutant **un par un** les éléments, c’est **d’abord colonne, puis ligne** (cf à gauche)

**Nb GridPane :**

Lors du développement, on peut être amené à devoir afficher la grille d'un GridPane de manière à faciliter le test ou le débogage. Pour cela, il faut modifier la propriété **gridLinesVisible** de la classe GridPane et lui donner une valeur égale à **true**.

root.setGridLinesVisible(true);

**TP 1 :**

On cherche à construire un panneau, on le décompose en un borderPane, dans lequel les 4 boutons sont dans une Hbox, les 4 labels en colonne dans une Vbox, et un pane pour utiliser le reste de l’espace avec le dessin.

De plus, on va diviser en classes pour créer l’outil : 5 classes, dans un nouveau package

**MainApp :**

**package** fr.afcepf.panneauObjet;

**import** com.sun.scenario.effect.impl.prism.PrImage;

**import** javafx.application.Application;

**import** javafx.scene.Scene;

**import** javafx.stage.Stage;

**public** **class** MainApp **extends** Application

{

@Override

**public** **void** start(Stage primaryStage) **throws** Exception

{

MainPannel root = **new** MainPannel();

Scene scene = **new** Scene(root, 600, 400);

primaryStage.setScene(scene);

primaryStage.sizeToScene();

primaryStage.setResizable(**false**);

primaryStage.setTitle("C'est mon titre à moi.");

primaryStage.show();

// **TODO** Auto-generated method stub

}

**public** **static** **void** main(String[] args)

{

*launch*(args);

}

}

**MainPannel :**

**package fr.afcepf.panneauObjet;**

**import** javafx.scene.layout.BorderPane;

**public** **class** MainPannel **extends** BorderPane

{

**private** ButtonPannel buttonPannel = **new** ButtonPannel();

**private** LabelPannel labelPannel = **new** LabelPannel();

**private** CenterPannel centerPannel = **new** CenterPannel();

**public** MainPannel()

{

**super**();

setTop(buttonPannel);

setLeft(labelPannel);

setCenter(centerPannel);

/// A TERMIENR

}

}

**ButtonPannel :**

**package fr.afcepf.panneauObjet;**

**import** javafx.geometry.Pos;

**import** javafx.scene.control.Button;

**import** javafx.scene.layout.HBox;

**public** **class** ButtonPannel **extends** HBox

{

**private** Button btn1 = **new** Button("Bouton 1");

**private** Button btn2 = **new** Button("Bouton 2");

**private** Button btn3 = **new** Button("Bouton 3");

**private** Button btn4 = **new** Button("Bouton 4");

// classe de panneaux objets, que l'on va pouvoir utiliser à foison

**public** ButtonPannel()

{

**super**();

**this**.getChildren().addAll(btn1, btn2, btn3, btn4);

**this**.setSpacing(50);

**this**.setAlignment(Pos.***CENTER***);

**this**.setStyle("-fx-background-color: grey; -fx-border-color: hotpink");

**this**.setPrefSize(400, 75);

}

}

**LabelPannel :**

**package fr.afcepf.panneauObjet;**

**import** javafx.geometry.Pos;

**import** javafx.scene.control.Label;

**import** javafx.scene.layout.VBox;

// extends VBox car les labels sont dans une VBOx

**public** **class** LabelPannel **extends** VBox

{

**private** Label lbl1 = **new** Label("Babibelle 1");

**private** Label lbl2 = **new** Label("Paul Kalkerbrenner");

**private** Label lbl3 = **new** Label("Moby");

**private** Label lbl4 = **new** Label("La vache qui pleure.");

// constructeur (CTRL MAJ S)

**public** LabelPannel()

{

**super**();

**this**.getChildren().addAll(lbl1, lbl2, lbl3, lbl4);

**this**.setSpacing(50);

**this**.setAlignment(Pos.***CENTER***);

**this**.setStyle("-fx-background-color: grey; -fx-border-color: hotpink");

**this**.setPrefSize(100, 200);

}

}

**CenterPannel**:

**package fr.afcepf.panneauObjet;**

**import** javafx.scene.layout.Pane;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Circle;

**public** **class** CenterPannel **extends** Pane

{

**private** Circle cercle = **new** Circle(50);

**public** CenterPannel()

{

**super**();

**this**.getChildren().add(cercle);

cercle.setFill(Color.***BROWN***);

**this**.setStyle("-fx-backkground-color: green");

**this**.setPrefSize(1000, 1000);

cercle.relocate(75,75);

}

}

**Les événements**

Astuce pour autoriser la validation via ENTER





**package fr.afcepf.evenement;**

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.HBox;

**import** javafx.stage.Stage;

**public** **class** PremierEvenement **extends** Application

{

@Override

**public** **void** start(Stage primaryStage) **throws** Exception

{

HBox root = **new** HBox(20);

root.setAlignment(Pos.***CENTER***);

TextField tf = **new** TextField();

Label lbl = **new** Label("No texte, pédé.");

Button btn = **new** Button("Clic pédé");

// accepter que la touche ENTRER valide

btn.setDefaultButton(**true**);

// ajout des bébés

root.getChildren().addAll(btn, tf, lbl);

// on veut qu'il se passe des choses quand on clique sur le bouton, pédé.

// on verra ensuite les listeners, pédé.

btn.setOnAction(**new** EventHandler<ActionEvent>()

{

@Override

**public** **void** handle(ActionEvent event)

{

// System.out.println(tf.getText());

// root.setStyle("-fx-background-color: hotpink");

lbl.setText(tf.getText());

}

});

Scene scene = **new** Scene(root, 400, 100);

primaryStage.setScene(scene);

primaryStage.sizeToScene();

primaryStage.setTitle("Titre n° 14, pédé");

primaryStage.show();

}

**public** **static** **void** main(String[] args)

{

*launch*(args);

}

}

TP 2 :

Un panneau avec un panneau qui change de couleur en fonction du clic sur les boutons en haut avec les mots bleu, rouge, hotpink, orange.

On fait un panneau principal dans lequel sont tous les autres objets,  
On peut ainsi utiliser les geter seters pour récupérer / donner les couleurs.

On peut gérer les événements de quatre manières différentes :  
-  
- dans le panneau principal

- dans le constructeur di panneau qui contient la source de l’événement ; on récupère l’instance du pannel root en remontant l’arbre des composants jusqu’à la scène puis avec la méthode GetRoot() on récupère la racine de la scène.

- avec une classe qui gère l’événement (qui implements EventHandler<ActionEvent>)

Voir ici OCR [clic](https://openclassrooms.com/courses/les-applications-web-avec-javafx/les-pre-requis-1)

**TP2 – Main App**

**package tpEventObjet;**

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.Scene;

**import** javafx.stage.Stage;

**public** **class** MainApp **extends** Application{

@Override

**public** **void** start(Stage stage) **throws** Exception {

**final** MainPanel root = **new** MainPanel();

Scene scene = **new** Scene(root);

stage.setScene(scene);

stage.show();

//Premiere methode pour gerer un evenement

//en objet: dans la MainApp

root.getButtonsPanel().getRedButton()

.setOnAction(**new** EventHandler<ActionEvent>() {

@Override

**public** **void** handle(ActionEvent event) {

root.getColorPanel()

.setStyle("-fx-background-color: red");

}

});

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

**TP2 - Main Pannel**

**package tpEventObjet;**

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.layout.GridPane;

**public** **class** MainPanel **extends** GridPane {

**private** ButtonsPanel buttonsPanel = **new** ButtonsPanel();

**private** ColorPanel colorPanel = **new** ColorPanel();

**public** MainPanel() {

addColumn(0, buttonsPanel, colorPanel);

OrangeButtonHandler handler

= **new** OrangeButtonHandler(colorPanel);

buttonsPanel.getOrangeButton().setOnAction(handler);

//Deuxieme facon de gerer un evenement:

//dans le panneau principal

buttonsPanel.getBlueButton()

.setOnAction(**new** EventHandler<ActionEvent>() {

@Override

**public** **void** handle(ActionEvent event) {

colorPanel.setStyle("-fx-background-color: blue");

}

});

}

**public** ButtonsPanel getButtonsPanel() {

**return** buttonsPanel;

}

**public** ColorPanel getColorPanel() {

**return** colorPanel;

}

}

**TP2 - Color Pannel  
  
package** tpEventObjet;

**import** javafx.scene.layout.FlowPane;

**public** **class** ColorPanel **extends** FlowPane {

**public** ColorPanel() {

setPrefSize(200, 200);

setStyle("-fx-background-color: yellow");

}

}

**TP2 – ButtonsPanel   
 Avec des geter seter**

**package** tpEventObjet;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Insets;

**import** javafx.geometry.Pos;

**import** javafx.scene.control.Button;

**import** javafx.scene.layout.HBox;

**public** **class** ButtonsPanel **extends** HBox {

**private** Button redButton = **new** Button("Rouge");

**private** Button blueButton = **new** Button("Bleu");

**private** Button greenButton = **new** Button("Vert");

**private** Button orangeButton = **new** Button("Orange");

**public** ButtonsPanel() {

getChildren().addAll(redButton, blueButton, greenButton, orangeButton);

setSpacing(5);

setAlignment(Pos.***CENTER***);

setPadding(**new** Insets(5));

//Troisieme moyen de gerer un evenement: dans le constructeur

//du panneau qui contien la source de l'evenement.

greenButton.setOnAction(**new** EventHandler<ActionEvent>() {

@Override

**public** **void** handle(ActionEvent event) {

//On recupere l'instance de mon pannel root

//en remontant l'arbre de mes composants jusqu'Ã  la scene

//puis, avec la methode getRoot(), je recupere la racine

//de ma scene

MainPanel root =

(MainPanel) ButtonsPanel.**this**.getScene().getRoot();

root.getColorPanel().setStyle("-fx-background-color : green");

}

});

}

**public** Button getRedButton() {

**return** redButton;

}

**public** Button getBlueButton() {

**return** blueButton;

}

**public** Button getGreenButton() {

**return** greenButton;

}

**public** Button getOrangeButton() {

**return** orangeButton;

}

}

**TP2 – OrangeButtonHandler**

**public** **class** OrangeButtonHandler **implements** EventHandler<ActionEvent>{

**private** ColorPanel colorPanel;

**public** OrangeButtonHandler(ColorPanel colorPanel)

{

**this**.colorPanel = colorPanel;

}

@Override

**public** **void** handle(ActionEvent event)

{

colorPanel.setStyle("-fx-background-color: orange");

}

}

**TP Calculatrice :**

Prof « j’ai déclaré mes variables private, tableau de tableau de strings :

String[][] buttons »

Besoin aussi d’un textField « affichage », valeur par défaut 0.  
Bouton clear, aussi.

On a tout, on peut commencer à aller dans la méthode Start.  
On y instancie un borderPane (root), une scene (scene),  
On gère la primaryStage (sizeToScene, setScene, titre, et show())

On a alors le layout de fond, on peut commencer à construire (dans Start toujours).

Configuration du panneau d’affichage du résultat :  
On part d’une Hbox qui sera le top : Hbox top = new Hbox.

Top.setAlignment(pos.CENTER)

On veut que l’affichage ne soit pas trop grand, donc on le set à 270 :  
affichage.setPrefWidth(270)

Idem pour son alignement, on le veut bien centré : affichage.setAlignment(pos.CENTER)  
Idem bouton clear : clear.setAlignement(Pos.CENTER)

On veut donner à la Hbox tous ces éléments qu l’on vient de créer et de styliser : top.getChildren().addAll(affichage, clear)

On donne a root la box : root.setTop(top)

On veut insérer dans une Vbox des Hbox successives pour y faire les boutons.

Configuration du panneau de boutons: ce sera donc une Vbox

Vbox centre = new Vbox

on set son alignement : centre.setAlignment(pos.CENTER)

On va itérer pour créer les boutons, c’est l’intérêt de la déclaration du tableau de strings !

Pour chaque tableau de Strings (nommé string), on instancie une Hbox !

On instancie donc une premiere Hbox dans la boucle :  
Hbox hb = new Hbox

Pour chaque string présent dans le tableau i, on instancie alors des boutons (deuxième boucle)

Button btn = new Button(string) ;

Stylisation : btn.setStyle(«-fx-font-size : 25px »)  
btn.setPrefSize(80,80)

On donne les boutons à la Hbox : hb.getChildren().add(btn)

Avant de changer de tableau String, on add la Hbox à la Vbox : centre.getChildren().add(hb)

Attention, on ne pourra pas faire de setonAction puisque le bouton n’est pas récupérable ensuite (variable locale). On utilisera un autre moyen, c’est aussi l’intérêt du TP.

On a un soucis à régler, on veut que le dernier bouton (le =) soit deux fois plus gros que les autres, horizontalement ; il faut récupérer cette valeur.

On récupère d’abord la dernière liste (ligne de code intéressante ) :

Hbox last = (Hbox) centre.getChildren().get(centre.getChildren().size-1)

On remarque un castage en Hbox.

On réitère dans cette Hbox pour identifier le dernier bouton :

Button lastBtn = (button) last.getChildren().get(last.getChildren().size()-1)

On peut enfin modifier sa largeur : lastBtn.setPrefWidth(160), son style, etc.

On a donc tous les boutons, on set la Vbox au centre du borderPanel:

root.setCenter(centre) → on donne en panneau central la Vbox Centre

On veut maintenant gérer les événements, et cela dans la même classe (car on est sur un projet basique).

On rajoute donc l’implementation de EventHandler<ActionEvent> à la classe CalculatriceBasique, en plus de l’extend Application déjà présent.

Quels événements veut-on gérer ?

L’événement du bouton clear :  
Clear.setOnAction(this) → prend le eventhandler qui est dans la classe, et pas à l’extérieur.

Même chose pour les boutons : on doit leur dire btn.setOnAction(this) → dans la même méthode Handle on devra doncc gérer les actions de TOUS les boutons !

On déclare pour cela un bouton dans la méthode handle :  
Button btnClick = (Button) event.getSource() → avec castage en Bouton.

Cette méthode getSource est pratique, elle permet à partir de l’event GetSource savoir quel est le bouton qui a déclanché l’événement !!

On va donc pouvoir mettre ensuite des conditions, qui vont influencer l’événement qui doit en découler.

Comment différentier les boutons qui ont des valeurs numériques, et ceux qui n’en ont pas ?  
On tente de les parseInt : selon le résultat du try catch, on aura la le résultat.

Integer.parseInt(btnClick.getText())

Affichage.setText(affichage.getText() + btnclick.getText())

A ce stade, ça marche mais aucune exception n’est catchée si val bouton non numérique cliqué,  
On crée donc le try catch avec une exception e (cache misère car n’explique rien, sauf qu’il y a une exception).

Si l’on est dans le catch, on doit gérer les autres boutons, les non numériques : on crée un switch case sur le label du bouton (btnClick.getText() ).

On doit gérer les cas + , -, \*, /, (, ), .

Simplement :  
affichage.setText(affichage.getText() + btnCLick.getText()) ;break ;

Pour le cas Clear (C) : affichage.setText(« 0 »);break ;

Cas du « +/- » :  
If(affichage.getText().indexOf(‘-’) == 0) → on doit changer le signe.

InfexOf renvoie en effet le positionnement de ‘-’ si le signe existe,  
On va vérifier si ce signe est en première position, auquel cas oui, on doit le changer. Si renvoie 0, donc !

Reprenons pour « changer » le signe, en fait on l’efface :

If(affichage.getText().indexOf(‘-’){

affichage.setText(affichage.getText().substring(1)) ;}

On récupère la chaine identique, à partir d’un indice décalé de 1 !

Sinon, else :  
affichage.setText(« - » + affichage.getText()) ;

Il nous reste le case « = » :  
On utilise un script, donc une partie de code cadeau gifto.

D’autre part, on peut enlever le 0 qui est au début de tout calcul, et reste avant quand on clic des boutons numériques, en rajoutant :

if (affichage.getText().equals(« 0 »))

{

affichage.setText(« ») ; → chaine vide}

Autre petit truc à gérer : quand on a fait égal, que la prochaine touche reparte de 0 et pas du précédent résultat.  
Pour cela, ajout d’un boolean (lastClicEgal), et au tout début du handle on gère cela :  
if (lastClicEgal) → affichage.setText(«»)

**Prof :** intéressant car montre qu’on peut faire une collection (liste ou tableau d’objets, puis itérer sur liste ou tableau d’objets)  
De plus, possibilité de gérer les événements au même endroit, et différer les traitements à partir de la source.

Important aussi la façon dont est gérée le if avec les break.

**TP – CALCULATRICE – CalculatriceBasique**

**package** fr.afcepf.tpcalculatrice;

**import** javax.script.ScriptEngine;

**import** javax.script.ScriptEngineManager;

**import** javax.script.ScriptException;

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.BorderPane;

**import** javafx.scene.layout.HBox;

**import** javafx.scene.layout.VBox;

**import** javafx.stage.Stage;

**public** **class** CalculatriceBasique **extends** Application **implements** EventHandler<ActionEvent>{

**private** String[][] buttons = {

{"1", "2", "3", "+"},

{"4", "5", "6", "-"},

{"7", "8", "9", "/"},

{"+/-", "0", ".", "\*"},

{"(", ")", "="}

};

**private** TextField affichage = **new** TextField("0");

**private** Button clear = **new** Button("C");

**private** **boolean** lastClickEgal = **false**;

@Override

**public** **void** start(Stage primaryStage) **throws** Exception

{

BorderPane root = **new** BorderPane();

Scene scene = **new** Scene(root, 320, 420);

//Configuration du panneau d'affichage du resultat:

HBox top = **new** HBox();

top.setAlignment(Pos.***CENTER***);

affichage.setPrefWidth(270);

affichage.setAlignment(Pos.***CENTER***);

clear.setAlignment(Pos.***CENTER***);

clear.setPrefWidth(50);

clear.setOnAction(**this**);

top.getChildren().addAll(affichage, clear);

root.setTop(top);

//Configuration du panneau de boutons

VBox centre = **new** VBox();

centre.setAlignment(Pos.***CENTER***);

**for** (String[] strings : buttons) {

HBox hb = **new** HBox();

**for** (String string : strings) {

Button btn = **new** Button(string);

btn.setStyle("-fx-font-size : 25px");

btn.setPrefSize(80, 80);

btn.setOnAction(**this**);

hb.getChildren().add(btn);

}

centre.getChildren().add(hb);

}

//Je recupere le dernier enfant de ma VBox, cad la HBox qui contient le bouton egal:

HBox last = (HBox) centre.getChildren().get(centre.getChildren().size()-1);

//Je recupere le derniere enfant de ma VBox,, cad le bouton "egal" que je souhaite modifier:

Button lastBtn = (Button)last.getChildren().get(last.getChildren().size() - 1);

lastBtn.setPrefWidth(160);

lastBtn.setStyle("-fx-font-size: 35px");

root.setCenter(centre);

primaryStage.setScene(scene);

primaryStage.sizeToScene();

primaryStage.setTitle("Calculatrice Basique");

primaryStage.show();

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

@Override

**public** **void** handle(ActionEvent event)

{

Button btnClick = (Button) event.getSource();

**if**(affichage.getText().equals("0")

{

affichage.setText("");

}

**if** (lastClickEgal)

{

affichage.setText("");

}

**try**

{

Integer.*parseInt*(btnClick.getText());

affichage.setText(affichage.getText() + btnClick.getText());

lastClickEgal = **false**;

}

**catch** (Exception e)

{

**switch** (btnClick.getText())

{

**case** "+":

**case** "-":

**case** "\*":

**case** "/":

**case** "(":

**case** ")":

**case** ".":

affichage.setText(affichage.getText() + btnClick.getText());

lastClickEgal = **false**;

**break**;

**case** "C":

affichage.setText("0");

lastClickEgal = **false**;

**break**;

**case** "+/-":

**if**(affichage.getText().indexOf('-') == 0){

affichage.setText(affichage.getText().substring(1));

}**else**{

affichage.setText("-" + affichage.getText());

}

lastClickEgal = **false**;

**break**;

**case** "=":

ScriptEngineManager manager = **new** ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("js");

**try**

{

Object result = engine.eval(affichage.getText());

affichage.setText(result.toString());

lastClickEgal = **true**;

} **catch** (ScriptException ex)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**default**:

**break**;

}

}

}

}

**TP – CALCULATRICE – TestCalculatrice**

**package** fr.afcepf.tpcalculatrice;

**import** javax.script.ScriptEngine;

**import** javax.script.ScriptEngineManager;

**import** javax.script.ScriptException;

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.control.TextField;

**import** javafx.scene.layout.HBox;

**import** javafx.stage.Stage;

**public** **class** TestCalculatrice **extends** Application{

**public** **static** **void** main(String[] args) {

*launch*(args);

}

@Override

**public** **void** start(Stage primaryStage) **throws** Exception {

HBox root = **new** HBox(10);

Scene scene = **new** Scene(root);

primaryStage.setWidth(400);

primaryStage.setHeight(100);

root.setAlignment(Pos.***CENTER***);

TextField tf = **new** TextField();

Button btn = **new** Button("Calculer");

Label affichage = **new** Label("Pas de calcul");

root.getChildren().addAll(tf, btn, affichage);

btn.setOnAction(**new** EventHandler<ActionEvent>() {

@Override

**public** **void** handle(ActionEvent event) {

ScriptEngineManager manager = **new** ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("js");

**try** {

Object result = engine.eval(tf.getText());

affichage.setText(result.toString());

} **catch** (ScriptException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

});

primaryStage.setScene(scene);

primaryStage.show();

}

}

On va voir comment lier deux scènes.

Stage **stage = (Stage) FirstPan.this.getScene().getWindow(); // getWindow permet de récupérer le root**

**FileChooser :**

On veut que l’utilisateur puisse choisir un fichier, et que l’on récupère son chemin d’accès absolu.  
On a besoin d’un bouton (et d’un label),

btn.setOnAction avec EventHandler,  
Et dans public void Handle(ActionEvent event) on va gérer le FileChooser :

FileChooser fileChooser = new FileChooser() ;

On lui dit ensuite :

File f (avec import de la classe File de java.io) = fileChooser.showOpenDialog(primaryStage.getOwner()) ;

Cela ouvre une popup et primaryStage.getOwner() sert à préciser que ma popup doit appartenir au contexte de la primaryStage.

On récupère, mais on n’affiche pas encore,  
Il faut d’abord vérifier qu’on n’a pas u truc null :  
if (f!= null) alors lbl.setText(f.**getAbsolutPath()**)

On peut aussi faire un File initDir = new File(« C:/Users/Formation/Desktop»)  
Cela change évidemment selon l’ordi.

On peut ensuite faire le  
fileChoooser.setInititialDirectory(initDir)

Cela évidemment avant de demander à l’utilisateur avec le showOpenDialog.

On arrive ainsi directement sur le Desktop au moment de l’ouverture de la boite de dialogue.

**CheckBox**

Comme d’habitude, hérite de Application.

Méthode main : setup habituel → launch(args), en root une Vbox, une scène scene donnée au primaryStage, ce dernier ayant toutes les caract habituelles.

Création d’un premier label, puis de checkboxes (classe javaFx)

La CheckBox prend un argument un string (normal : texte explicatif).

Création de quatre cbx (cb1, cb2, cb3, cb4).

On crée un bouton « valider » : Button btn = new Button(« Valider ») ;

Mieux, on va créer un tabeau de checkboxes :  
CheckBox[] boxes = {cb1, cb2, cb3, cb4} ;

On ne peut ajouter avec AddAll car on n’a pas de nœuds.  
Cela dit, on peut aussi ajouter une collection donc d’abord add(lbl), puis addAll(boxes), et add(btn).

Pour récupérer les valeurs, on passe comme d’habitude par un SetOnAction :

SetOnAction(new EventHandler<ActionEvent>()

On a comme d’habitude la méthode créée automatiquement handle(ActionEvent event)

On y entre un StringBuilder :

StringBuilder  builder = new StringBuilder () ;

**boucle sur les boxes avec méthode .isSelected() qui renvoie un boolean.**

On utilise aussi : String.format(arg1, arg2) → on lui donne le format, puis la string à afficher.  
Pour le format, convention particuilère pour aller à la ligne : termine par %n  
Ainsi : **String.format**(«\t%s%n», checkbox.getText())

Si la checkbox est selectionnée, alors on builder.append(String.format)() :  
  
builder.append(String.format(«\t%s%n», checkbox.getText()));

Après le for, on va juste dire que si le builder.length() == 0, alors builder.append(« Rien sélectionné ! »).

Sinon, on affiche le builder :

builder.insert → permet d’ajouter au builder

builder.insert(0, «Vos personnages préférés sont : \n») ;

Affichage du builder : syso(builder).

On rajoute une TextArea ta sinon ça n’apparait qu’en console, un peu relou :  
TextArea ta = new TextArea(), puis ta.setText(builder.toString())

**RadioButton**

Classe : PremierRadioButton

On crée une Hbox, une Scene, un RadioButtonrb1 :  
RadioButton rb1 = new RadioButton(« Féminin ») ;  
RadioButton rb2 = new RadioButton(«Masculin ») ;

On crée aussi un bouton btn («Valider»)

On ajoute tous ces composants à root :  
root.getChildren.addAll(rb1, rb2, rb3, btn)

On centre la Hbox : root.setAlignment(pos.CENTER)

Comme on veut qu’un seul bouton soit coché, on va lier ces trois boutons entre eux : on utilise un toggleGroup.

ToggleGroup grp = new ToggleGroup()

On utilise la propriété getToggle pour ajouter au Toggle :

grp.getToggle().addAll(rb1, rb2, rb3)

btn.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

et ici on mt la Labeled itemSelected avec un cast

Labeled itemSelected = (Labeled) grp.getSelectedToggle() ;

**Le cast Labeled permet d’obtenir la méthode getText() sur nos radiobuttons**

On sysout notre itemSelected.getText()  
 sans passer par le cast, on ne peut aboutir car le toggle n’a pas la méthode getText

On récupère désormais la valeur grace à un listener :  
Ce dernier écoute la propriété d’un objet, on va ici vérifier si l’objet est sélectionné.

On va écouter le changement de propriété dans le groupe, et déclancher un événement dès que changement.

Ici, le groupe possède la propriété selectedToggleProperty.  
On va faire grp.selectedToggleProperty().addListener(new ChangeListener<>())

On veut listener un toggle ici, donc : ChangeListener<Toggle>

En découle la création automatique d’une public void changed.

ObservableValue < **?** extends Toggle> observable, Toggle oldValue, Toggle newValue

Label itemSelected = (Labeled) newValue  avec donc cast en labeled.

A chaque fois que l’utilisateur clic sur une case, on récupère la nouvelle valeur, on la rend imprimable grace au cast en labeled, puis on imprime.  
Sans le cast ça ne marche pas.

Le **?** désigne le fait qu’on se fou de savoir exactement quel objet est ici concerné.

Dès changement de valeur, on a désormais une impression en console de la nouvelle value.

On peut initialiser le Toggle avec :  
grp.selectToggle(rb1).

**ChoiceBox ou ComboBox**

**package fr.afcepf.composantAvecDonnees;**

**import** javafx.application.Application;

**import** javafx.collections.FXCollections;

**import** javafx.collections.ObservableList;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.ComboBox;

**import** javafx.scene.layout.HBox;

**import** javafx.stage.Stage;

**import** javafx.util.StringConverter;

**public** **class** AdvancedComboBox **extends** Application {

@Override

**public** **void** start(Stage primaryStage) **throws** Exception {

HBox root = **new** HBox(10);

Scene scene = **new** Scene(root, 500,100);

//J'instancie un dao pour avoir accÃ¨s Ã  ses mÃ©thodes de rÃ©cupÃ©ration de donnÃ©es

PersonageDao dao = **new** PersonageDao();

//J'instancie une liste observable Ã  laquelle je donne en argument la liste issue

//de la methode getAllPersonnages de mon dao.

ObservableList<Personnage> observablePersos = FXCollections.*observableArrayList*(dao.getAllPersonnages());

//J'instancie une combo box Ã  laquelle je donne en argument ma liste observable:

ComboBox<Personnage> cb = **new** ComboBox<>(observablePersos);

cb.setVisibleRowCount(2);

Button btn = **new** Button("Valider");

root.getChildren().addAll(cb, btn);

//Je paramÃ¨tre un convertir qui redifini l'affichage des mes objets dans la combobox:

cb.setConverter(**new** StringConverter<Personnage>() {

@Override

**public** String toString(Personnage perso) {

**return** String.*format*("%s %s", perso.getPrenom(), perso.getNom());

}

@Override

**public** Personnage fromString(String string) {

// **TODO** Auto-generated method stub

**return** **null**;

}

});

btn.setOnAction(**new** EventHandler<ActionEvent>() {

@Override

**public** **void** handle(ActionEvent event) {

Personnage persoSelected = cb.getSelectionModel().getSelectedItem();

System.***out***.println(persoSelected);

}

});

primaryStage.setScene(scene);

primaryStage.sizeToScene();

primaryStage.setTitle("Advances Comco Box");

primaryStage.show();

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

ChoiceBox qui extend Application, un Hbox root, une Scene scene.

On peut faire une ChoiceBox d’objets complets, mais nous commençons avec des String simples :

ChoiceBox<String> cb = new ChoiceBox<>() ;

Ajout d’un bouton valider

On ajoute les deux a root : getChildren add all etc.

Comment rajouter des éléments à la choiceBox ?  
GetItem obersvable

On rempli d’une liste manuellement la checkbox : cb.getItems().addAll(« Villes », « Paris », « Ouagadougou », « NY ») ;

On force la sélection du premier item à l’initialisation :

cb.getSelectionModel().selected(0) ;

bouton valider (btn)

btn.setOnAction

Dans la void handle :

**String itemSelected = cb.getSelectionModel().getSelectedItem()**

Cette ligne permet de récupérer l’item sélectionné, le print ensuite ou autre.

Pour le listener, on passe par  
**cb.getSelectionModel().selectedItemProperty()**

cela permet d’écouter la propriété,  
On addListener(new ChangeListener<String>()

{ méthode Changed{} générée automatiquement})

On system.out.print**f**  pour print format :

system.out.print**f**(«%s =>%s%n », odlValue, newValue) ;

La partie de gauche : variables le plus souvent un pourcentage et une lettre, la lettre correspond à int (d), à string (s), etc

Donc ici, c’est « imprime moi une string que je te précise après, puis la flèche, puis la deuxième string que je te précise après, et retour ligne stp ».

**JavaFX**

Architecture MVVM (Model View / View Model)

View model

Model

View

Base de

données

.bin

.csv

command

Update, erase, create

affiche

upload

**Le modèle a besoin d’entités, et de DAO.**  
  
Les **entités** sont les objets métiers (entities), qui servent à récupérer des données dans une bdd, puis créer un objet Java qui puisse être utilisé dans mon programme.

Les **DAO** sont les Data Access Object : ce sont des objets d’accès aux données,  
Ils se servent des entités pour récupérer les objets présents (dans fichier ou dans base de données), pour les transformer en objet Java qui puissent être utilisés dans un programme.  
Ex Monopoly : « envoie-moi toutes les rues », il instancie les objets, et les ajoute à une liste.

Au niveau de la **VIEW**, on a l’affichage : c’est tout ce qui est accessible par l’utilisateur. IHM : Interface homme/machine.

Le **VIEW MODEL**: observable. On récupère les données, on les transforme en objets java, on les charge et les transmet à cette iste observable, pour ainsi pouvoir les transmettre à l’application et les afficher.

a

L’utilisateur est en contact avec la vue.

JavaFx permet de récupérer des données, et pas de n’en prendre qu’en dur.  
C’est l’intérêt, on crée donc AdvancedComboBox

On veut charger dans une Hbox (par exemple) un objet entier (plus seulement quelques critères).

**package fr.afcepf.composantAvecDonnees;**

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** PersonageDao {

**public** List<Personnage> getAllPersonnages(){

List<Personnage> persos = **new** ArrayList<>();

Personnage kurt = **new** Personnage("Cobain", "Kurt", 27);

Personnage freddy = **new** Personnage("Mercury", "Freddy", 45);

Personnage bob = **new** Personnage("Marley", "Bob", 36);

Personnage amy = **new** Personnage("Winehouse", "Amy", 27);

Personnage michel = **new** Personnage("Sardou", "Michel", 70);

Collections.*addAll*(persos, kurt, freddy, bob, amy, michel);

**return** persos;

}

}

Création d’un PersonnageDao dao = new PersonnageDao() ;

On crée une **observableList<Personnage> observablePersos = FXCollections.observableArrayList(dao.getAllPersonnages()) ;**

On ajoute une collection à la cbbox cb :

ComboBox<Personnage> cb = new Combobox<observablePersos>

La différence entre ChoiceBox et ComboBox : la première est censé ne pas avoir trop trop de choix.  
La combobox va avoir une méthode .setVisibleRowCount(n) → permet de n’afficher que n lignes à la fois.

On veut changer l’affichage de la combobox pour que ce soit plus classe,  
Pour cela on utilise un **converter**!

cb.setConverter(new StringConverter())

On peut ainsi définir ce qui va s’afficher dans la cbbox.

La méthode toString alors générée automatiquement doit être paramétrée pour retouner un Sring.formatée :

**return String.format(«%s %s âgé de %d », perso.getPrenom(), perso.getNom(), perso.getAge()) ;**

Normalement l’affichage de la box est alors plus présentable : la méthode SetConverter permet donc d’afficher un string plus beau que par défaut.

Ensuite on s’occupe des événements, avec le btn.setOnAction → on veut récupérer un personnage entier, d’où le :  
Personnage persoSelected = cb.getSelectionModel().getSelectedItem() ;

On peut ensuite sysout persoSelected.

On a des objets qui sont instanciés, à partir du DAO on met les objets en mémoire,  
Au niveau de l’intérmédiaire on a des listes observables hydratées de données grace au DAO, et ces listes on peut les donner aux composants IHM.

On a utilisé **FXCOllections.observableArrayList** pour instancier une liste observable, à partir de la liste issur de la méthode getAllPersonnages du DAO.

On a ensuite instancié une combobox à laqulle on a donné en argument la liste observable.

De fait, on n’affiche que certaines propriétés pour l’uilisateur, mais l’objet entier demeure en mémoire.

**TableView**

**package fr.afcepf.composantAvecDonnees;**

**import** javafx.application.Application;

**import** javafx.beans.value.ChangeListener;

**import** javafx.beans.value.ObservableValue;

**import** javafx.collections.FXCollections;

**import** javafx.collections.ObservableList;

**import** javafx.scene.Scene;

**import** javafx.scene.control.TableColumn;

**import** javafx.scene.control.TableView;

**import** javafx.scene.control.cell.PropertyValueFactory;

**import** javafx.scene.layout.AnchorPane;

**import** javafx.stage.Stage;

**public** **class** PremiereTableView **extends** Application {

@SuppressWarnings("unchecked")

@Override

**public** **void** start(Stage primaryStage) **throws** Exception {

AnchorPane root = **new** AnchorPane();

PersonageDao dao = **new** PersonageDao();

ObservableList<Personnage> observablePersos = FXCollections.*observableArrayList*(dao.getAllPersonnages());

TableView<Personnage> tableView = **new** TableView<>(observablePersos);

TableColumn<Personnage, String> colNom = **new** TableColumn<>("Nom");

colNom.setCellValueFactory(**new** PropertyValueFactory<>("nom"));

TableColumn<Personnage, String> colPrenom = **new** TableColumn<>("Prenom");

colPrenom.setCellValueFactory(**new** PropertyValueFactory<>("prenom"));

TableColumn<Personnage, Integer> colAge = **new** TableColumn<>("Age");

colAge.setCellValueFactory(**new** PropertyValueFactory<>("age"));

tableView.getColumns().addAll(colNom, colPrenom, colAge);

//Equilibrer la dimension des colonnes:

tableView.setColumnResizePolicy(TableView.***CONSTRAINED\_RESIZE\_POLICY***);

tableView.getSelectionModel().selectedItemProperty().addListener(**new** ChangeListener<Personnage>() {

@Override

**public** **void** changed(ObservableValue<? **extends** Personnage> observable, Personnage oldValue,

Personnage newValue) {

System.***out***.println(newValue);

}

});

root.getChildren().add(tableView);

AnchorPane.*setTopAnchor*(tableView, 5.);

AnchorPane.*setBottomAnchor*(tableView, 5.);

AnchorPane.*setLeftAnchor*(tableView, 5.);

AnchorPane.*setRightAnchor*(tableView, 5.);

Scene scene = **new** Scene(root, 500, 500);

primaryStage.setScene(scene);

primaryStage.sizeToScene();

primaryStage.setTitle("Premier TableView");

primaryStage.show();

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

On crée cette fois une AnchorPane en tant que root, sinon tout usuel.  
Titre : « premier table view ».

La TableView est une sorte de tableau hydratable en données.

On a besoin d’un dao : PersonnageDao dao = new PersonnageDao();

On need aussi comme juste avant une FXCollections.observableArrayList avec en paramètre dao.getAllPersonnages() :

FXCollections.observableArrayList(dao.getAllPersonnages())

On instancie de plus un TableView de personnages que l’on nomme tableView et l’on donne ObservablePersos en arg :

TableView<Personnage> tableView = new TableView<>(observablePersos) ;

De plus on root.getChildren().add(tableView)

TableColumnA ce stade en lançant, on n’a un tableau vide : normal, on n’a rien demandé d’afficher.  
On instancie donc une colonne (une tableColonne) :

TableColumn<quel est l’objet, quel est le type de l’attribut de l’objet que l’on veut afficher>

Ainsi ici :

TableColumn<Personnage, String> colNom = new TableColumn<>(« Nom ») ;

ColNom.**setCellValueFactory**(new PropertyValueFactory<>(« nom »))

… blabla de l’objet personnage qui porte le nom « nom ».

Dernière étape : faire le lien entre la colonne instanciée, et le header.

Pour cela, on fait un tableView.getColumns().add(colNom)

Ensuite on copie-colle le bout de paragraphe adéquat, pour remplir les colonnes Prénom, et Âge.

Finalement, on change la ligne du dessus en :

tableView.getColumns().add(colNom, colPrenom, colAge)

Dernière chose, petite propriété du tableview pour équilibrer la dimension des colonnes :tableView.setColumnResizePolicy(TableView.CONSTRAINED\_RESIZE\_POLICY)

Ajout d’un listener :

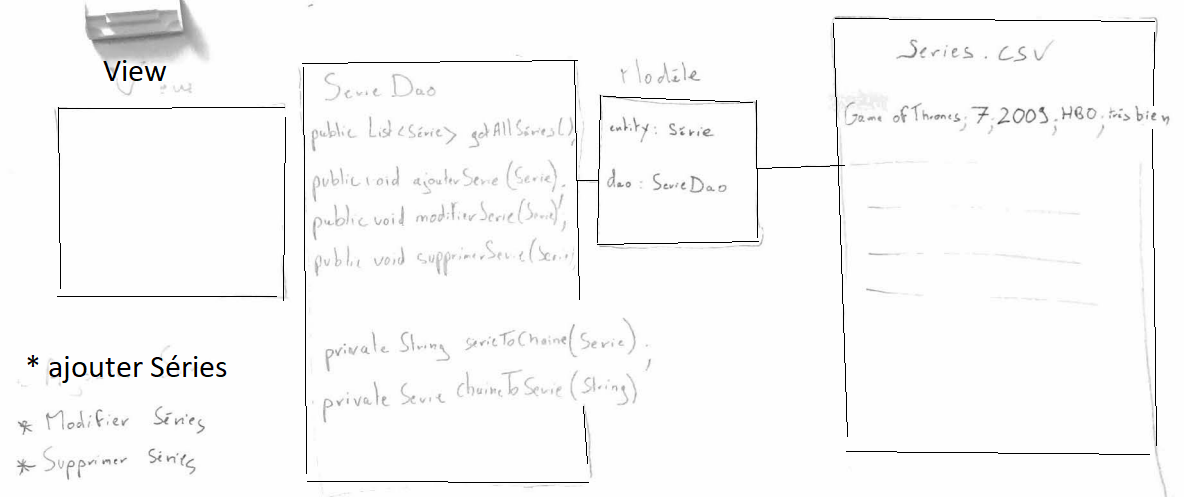
tableView.getSelectionModel().selectionItemProperty().addListener(new ChangeListener<Personnage>()

Du coup méthode changed développée automatiqument.

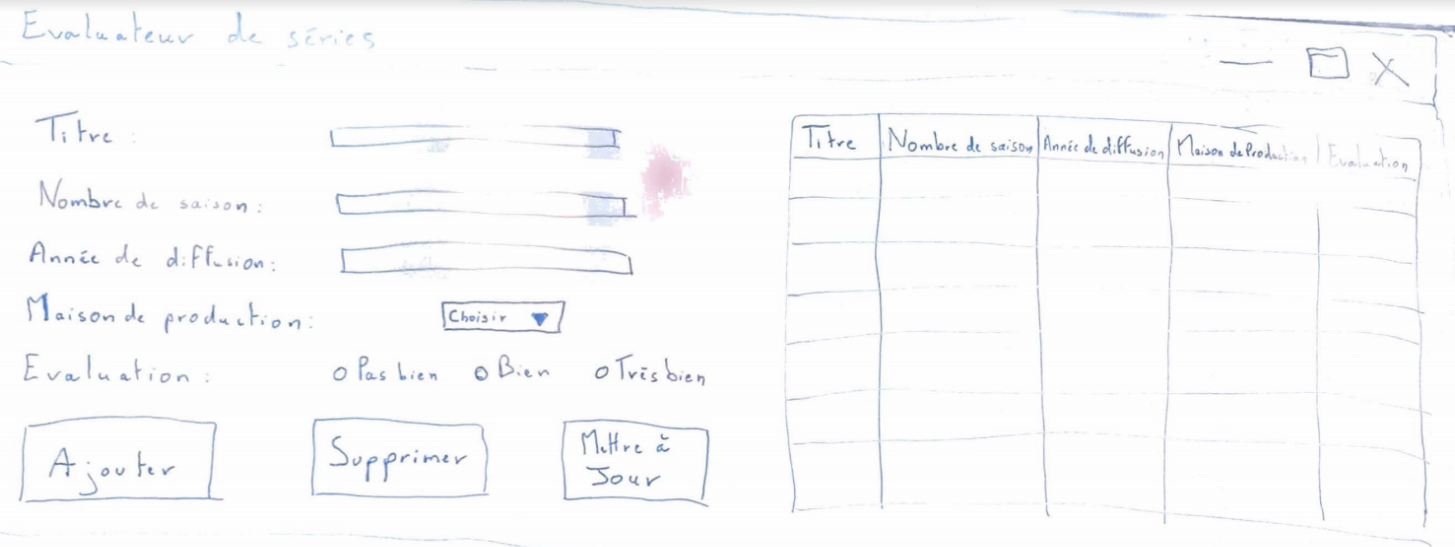
Ensuite, un sysout de la new value pour avoir le personnage sélectionné.

**TP les séries**

Fichier .csv organisé par éléments à la suite classés (Game of Thrones;7;2009;HBO;très bien → nomSerie;nombreSaisons;annéePremDiff;Diffuseur;Avis)



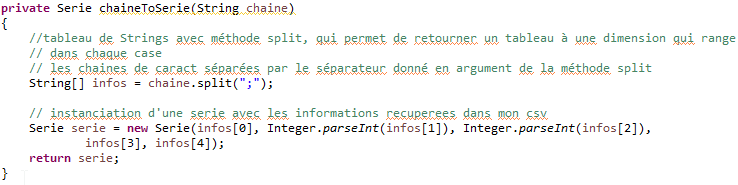
On veut pouvoir ajouter via l’outil dont l’interface est la suivante :



Que l’on puisse saisir les données puis les ajouter au tableau d’affichage à droite, et aussi au .csv

Astuces :



****

CTRL MAJ (-) pour fractionner le code en deux (ou remettre en un)

CTRL MAJ O pour importer les java.io ou autre  
CTRL -/+ pour (dé)zoomer

**Notes prof :**

Génération des geter seter, donc need des classes.

Il cre d’abord la classe **FormPanel** (avec constructeur) pour la partie de gauche, avec des label et textField associés (titre, nb saison, année de diff), aussi un label maison de prod et une choiceBox de Strings ChoiceBox<String> cbMaisonProd  
LabelEval pour « Evaluation : » et des radio button qui devront être liés,

Ensuite création d’un ToggleGroup « EvalGroup » pour pouvoir lier ces boutons

Une Hbox evalBox pour gérer tout ça.

Tout ça est passé en attributs private.

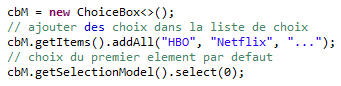
On ADD SHIFT S puis R pour les geter/seter, important : on choisit l’option de les mettre APRES le dernier truc qu’on a écrit, cette option est assez baba cool Raoul.

On revient au constructeur précédement généré,  
On instancie tout :  
lblTitre = new Label(« Titre : »)

txtTitre = new TextField() ;  
addRow(0, lblTitre, txtTitre) ;

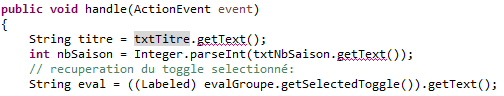
lblDiffusion = new Label(« Année de diffusion : ») ;

txtDiffusion = new TextField() ;  
AddRow(2, lblAnneeDif, txtAnneDif) ;

****ChoiceBox

Cours slides pdf Diderot : [clic](https://www.irif.fr/~zielonka/Enseignement/InterfaceGraphique/2016/SLIDES/COURS01/slides.pdf)

Prof : « je préfère faire la gestion des events du bouton là ou on a le bouton »,

****

Encore d’autres choses codées,

Attention à ce stade on peut ajouter, mais ne s’ajoute pas au .csv donc ne s’affiche plus sur l’outil si l’on ferme / rouvre !

Retour dans le SerieDao → on complète avec serieToChain(Serie serie)

Cette méthode private utilise un StringBuffer, et append les différentes rubriques avec des « ; » pour séparer dans le .csv

De plus, le buffer doit être re ToString à la fin (sinon n’est pas en chaine caract).