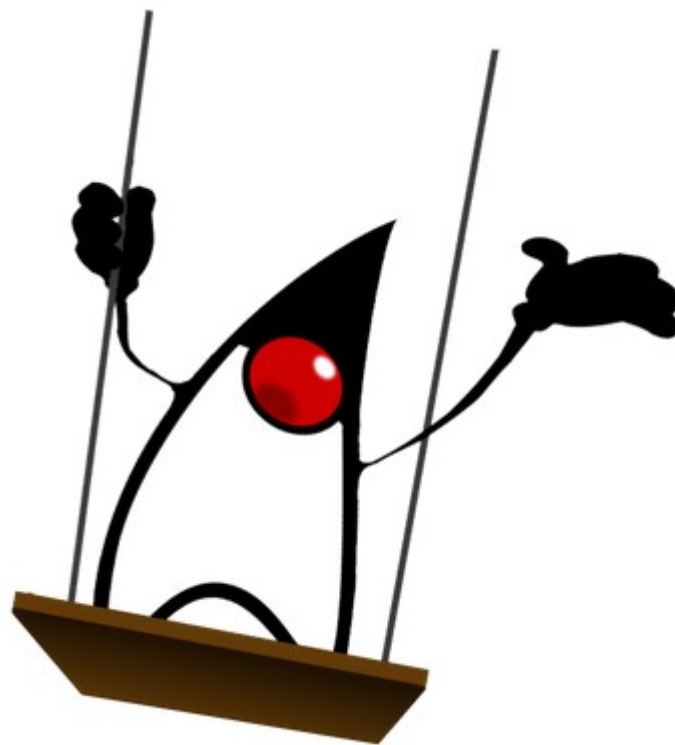


# Applications Desktop

## Initiation à Java FX



## TABLE DES MATIERES

<b>Section 1 C'est quoi une application Dektop ?</b>	<b>4</b>
1.1 Définition	4
1.2 Avantages et inconvénients	4
<b>Section 2 Introduction à Java FX</b>	<b>5</b>
2.1 Java FX	5
2.2 Télécharger JavaFX	5
2.3 Outil de développement	5
2.4 Liens web	5
2.5 Architecture	6
2.6 Créer une application simple	7
2.6.1 But	7
2.6.2 Créer le projet	8
2.6.3 Lancer l'application	8
2.6.4 Modifier le contenu de la fenêtre	9
2.6.5 Créer et appeler une nouvelle fenêtre	10
2.6.6 Ajouter du style	12
<b>Section 3 Retour sur la théorie</b>	<b>14</b>
3.1 Les conteneurs de Controls : javafx.scene.layout	14
3.2 Les Controls JavaFX	15
3.3 Les Styles avec CSS	16
3.3.1 Définition de styles globaux pour la scene	16
3.3.2 Définition de styles pour des composants identifiés	17
3.3.3 Faire connaître la feuille de style au fichier fxml	17
<b>Section 4 Application avec du style</b>	<b>18</b>
4.1 Créer le projet	18
4.2 Faire connaître un fichier de style	18
4.3 Placer les Layouts et Controls	19
4.4 Fenêtre de confirmation	21
<b>Section 5 Data Binding</b>	<b>22</b>
5.1 Binding unidirectionnel	22
5.2 Binding bi-directionnel	24
5.3 Placer un écouteur sur une propriété	24
<b>Section 6 utiliser un TableView</b>	<b>25</b>
6.1 But	25
6.2 Mode opératoire	25
6.2.1 Créer une liste de produits	27
6.2.2 Lier la liste à la TableView	27
6.2.3 Affichage du détail d'un produit	28
6.2.4 Modifier la liste des produits	28

<b><i>Section 7 La gestion des threads</i></b> .....	<b>29</b>
<b>7.1 Enoncé</b> .....	<b>29</b>
<b>7.2 Travail à faire</b> .....	<b>29</b>
<b>7.3 Résumé</b> .....	<b>31</b>

## Section 1 C'est quoi une application Dektop ?

---

### 1.1 Définition

<https://webtech.fr/blog/logiciel-de-bureau-vs-application-web-avantages-inconvenients/>

**Une application de bureau** (appelé aussi application native ou « bureau ») est un programme installé sur un poste de travail qui **ne dépend pas d'un navigateur pour fonctionner**. Une application de bureau est conçue spécifiquement à des fins prédéterminées, par exemple le traitement de texte Microsoft Word ou la suite bureautique iWork.

**Une application web** est une application qui fonctionne via un navigateur Web et n'a pas besoin d'être installée sur un ordinateur. Les applications Web sont généralement conçues pour être utilisées avec des systèmes d'exploitation et des appareils différents, ce qui permet aux utilisateurs de les utiliser à partir d'un ordinateur, d'une tablette ou d'un smartphone au lieu d'être liés à un seul type de poste de travail.

### 1.2 Avantages et inconvénients

Avantages d'une application de bureau :

- performance accrue car utilise directement l'OS local
- peut utiliser entièrement les ressources locales de la machine
- applications hautement personnalisées

Avantages d'une application web :

- pas d'installation en local côté utilisateur
- flexibilité et facilité d'emploi
- pas liée à l'OS du poste de l'utilisateur
- application disponible sur plusieurs supports, tablettes, tel ...

Inconvénients d'une application de bureau :

- doit être installée individuellement sur chaque poste
- les mises à jour sont aussi fastidieuses
- selon le code, peuvent être liées à un seul OS (code C, C++)

Inconvénient d'une application web :

- performances moindres
- ne peut pas gérer les ressources matérielles de l'OS

## Section 2 Introduction à Java FX

---

### 2.1 Java FX

**JavaFX** est une technologie créée par [Sun Microsystems](#) qui appartient désormais à [Oracle](#).

Avec l'apparition de Java 8 en mars 2014, JavaFX devient la bibliothèque de création d'interface graphique officielle du langage Java, pour toutes les sortes d'application (applications mobiles, applications sur poste de travail, applications Web), le développement de son prédécesseur [Swing](#) étant abandonné (sauf pour les corrections de bogues).

JavaFX contient des outils très divers, notamment pour les médias audio et vidéo, le graphisme 2D et 3D, la programmation Web, la programmation multi-fils etc.

Le SDK de JavaFX était intégré au JDK standard de Java 8 jusqu'à la version < 11, c'est maintenant un module indépendant

Nous allons développer avec l'outil IntelliJ, Netbeans étant trop compliqué à utiliser avec les versions de Java FX

### 2.2 Télécharger JavaFX

Utiliser le site <https://gluonhq.com/products/javafx/>

### 2.3 Outil de développement

L'outil IntelliJ est utilisé dans cette présentation.

Télécharger et installer la version Community  
<https://www.jetbrains.com/fr-fr/idea/download/#section=windows>

Le placement d'objets graphiques dans l'application va se faire avec l'outil scene builder. Il est intégré dans IntelliJ, moyennant quelques installations supplémentaires lors du 1<sup>er</sup> emploi.

L'outil peut aussi être utilisé en dehors de IntelliJ et offre dans ce cas plus d'espace pour dessiner. Scene builder est disponible à l'adresse  
<https://gluonhq.com/products/scene-builder/>

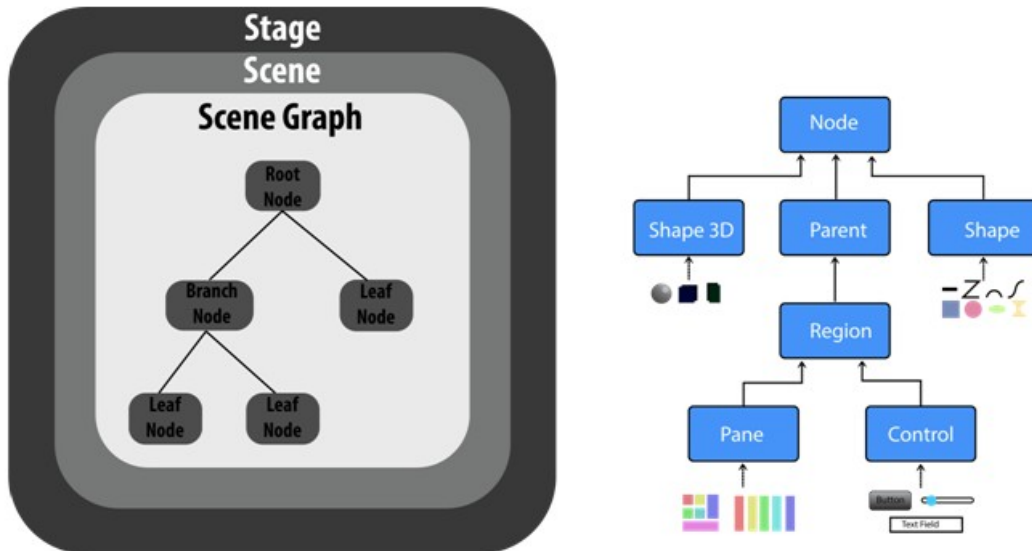
### 2.4 Liens web

[https://www.youtube.com/watch?v=pZzU-e\\_doNE](https://www.youtube.com/watch?v=pZzU-e_doNE)

<https://www.lirmm.fr/~fmichel/ens/java/cours/JavaFX.pdf>

## 2.5 Architecture

Une vue dans Java FX est un contenu arborescent.



L'extrait de code suivant contient les instructions pour définir une fenêtre avec un label :

```
Label unLabel = new Label("I'm a Label on new Window");

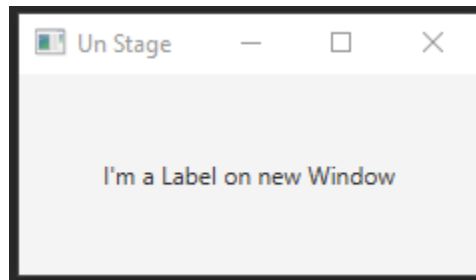
StackPane unLayout = new StackPane();
unLayout.getChildren().add(unLabel);

Scene uneScene = new Scene(unLayout, 230, 100);

// New window (Stage)
Stage newWindow = new Stage();
newWindow.setTitle("Un Stage");
newWindow.setScene(uneScene);

newWindow.show();
```

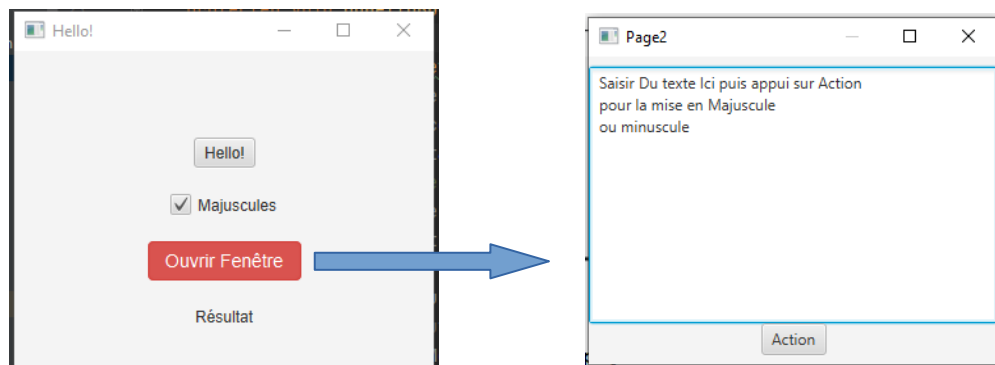
et affiche la fenêtre :



## 2.6 Créer une application simple

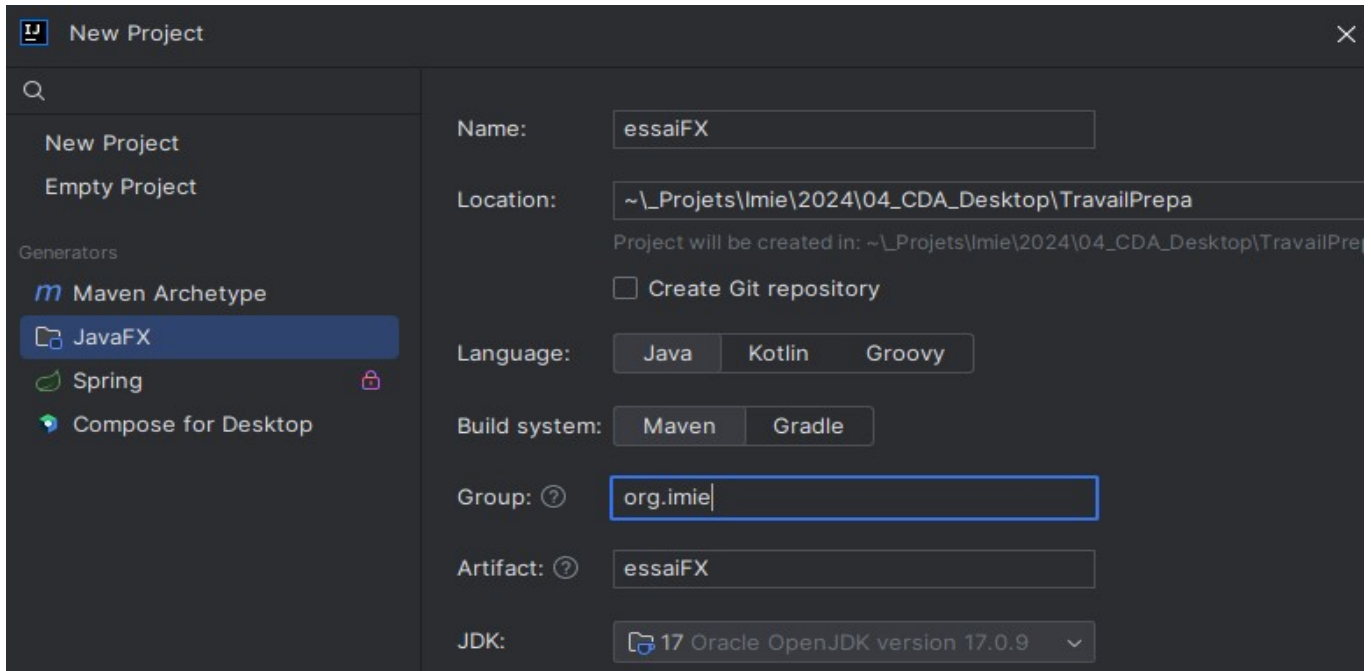
### 2.6.1 But

Nous allons créer une application de base avec 2 fenêtres. Il y a passage d'informations entre les fenêtres.

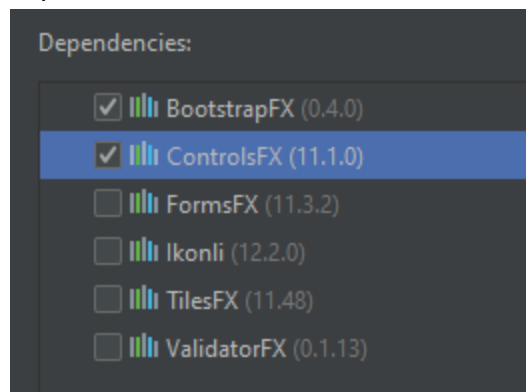


### 2.6.2 Créer le projet

Depuis IntelliJ, New Project puis saisir comme suit :



puis choisir les 2 dépendances suivantes :



Ces options sont visibles sur les sites :

<https://github.com/kordamp/bootstrapfx>

<https://controlsfx.github.io/>

### 2.6.3 Lancer l'application

Run sur l'application. Utiliser le bouton de la fenêtre.



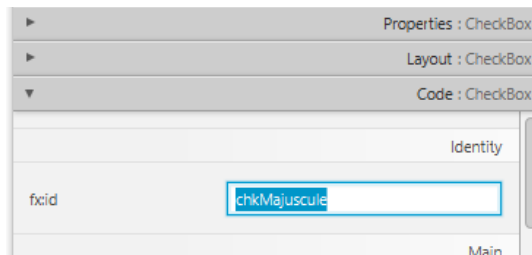
## 2.6.4 Modifier le contenu de la fenêtre

Editer le fichier hello-view.fxml.

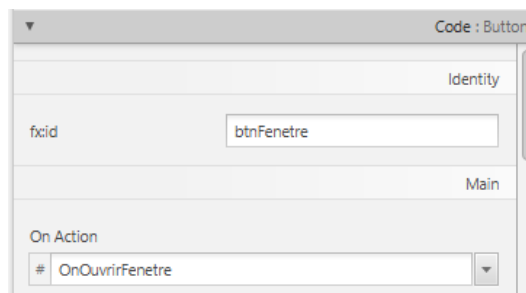
Clic sur l'onglet Scene Builder.

Ajouter sous le bouton existant :

- un checkbox. Le texte est « Majuscule », a pour identifiant chkMajuscule.

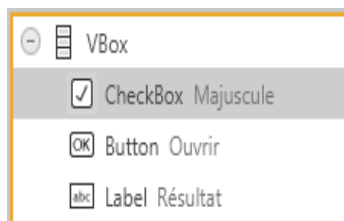


- Un bouton. Le texte est « Ouvrir Fenêtre ». L'identifiant est btnFenetre, l'action sur clic est OnOuvrirFenetre



- Un Label. Le texte est Résultat, L'Id est : lblResult

L'arborescence de vue doit ressembler à ceci :



La vue ressemble à ceci :



Revenir sur le source fxml.

Placer le curseur sur `OnOuvrirFenetre` et accepter le Create Method que propose IntelliJ

```
ng="false" text="Majuscules" />
false" onAction="#OnOuvrirFenetre" text="Ouvr
```

Placer le curseur sur `chkMajuscule` et demander de créer un attribut (create Field in Page2Controller). Idem pour `lblResult`.

Faire un Run de cette application.

### 2.6.5 Créer et appeler une nouvelle fenêtre

Le but est d'appeler une autre fenêtre sur appui du bouton `btnFenetre` de la fenêtre principale.

Nous allons de plus passer de l'information :

- depuis la fenêtre principale vers la nelle fenêtre : boolean qui indique si l'on souhaite traiter des majuscules ou non
- depuis la nelle fenêtre vers la fenêtre principale pour récupérer et afficher la chaîne transformée

Créer une nouvelle fenêtre :


Sur le nœud du projet `resources / com.imie.essaifx` faire New /FXML file. Donner pour nom `Page2`

Dans le fichier `Page2.fxml` indiquer le nom du contrôleur :

`fx:controller=" Page2Controller "`

Placer le curseur sur le mot `Page2Controller` et demander la création de la classe.

Ouvrir Scene Builder sur cette `Page2.fxml`

- Un  `AnchorPane` est placé comme élément par défaut dans la vue. Le remplacer par un `Vbox`
- Pour cela, clic droit sur `Anchor Pane` et choisir `Vrap in → VBox`
- Supprimer le `Anchor Pane`
- Ajouter un `TextArea` avec un id : `txtInput`
- Ajouter un `Bouton` avec le texte `Action`, id : `btnAction`, et le `On Action` : `OnAction`

Dans le fichier Page2Controller placer ce code (garder les lignes actuelles du haut)

```
public class Page2Controller {
    public TextArea txtInput;
    private boolean isMajuscule=false;

    public void setIsMajuscule(boolean isMajuscule) {
        this.isMajuscule = isMajuscule;
    }

    public void OnAction(ActionEvent actionEvent) {
        if (this.isMajuscule)
            txtInput.setText(txtInput.getText().toUpperCase());
        else
            txtInput.setText(txtInput.getText().toLowerCase());
    }
}
```

Analyser ce code.

Appeller cette vue.

L'appel se fait depuis la vue principale, dans HelloController.java

Placer le code suivant au bon endroit :

```
public void OnOuvrirFenetre(ActionEvent actionEvent) throws IOException {
    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("Page2.fxml"));
    Scene scene = new Scene(fxmlLoader.load(), 320, 240);
    Stage stage = new Stage();
    stage.setTitle("Page2");
    stage.setScene(scene);
    Page2Controller controller = fxmlLoader.getController();

    controller.setIsMajuscule(chkMajuscule.isSelected());
    //controller.txtInput.setText("ABCD");
    stage.initModality(Modality.APPLICATION_MODAL);
    stage.showAndWait();
    lblResult.setText(controller.txtInput.getText());
}
```

Analyser ce code.

Lancer l'application et faites des essais de transformation de texte.

Ici dans cet exemple minimal, les vues et objets graphiques ont été définis par fxml. Ils peuvent être définis aussi directement par du code java.

### 2.6.6 Ajouter du style

Nous avons fait le choix d'inclure BootstrapFX lors de la création du projet. Nous allons modifier la couleur du bouton btnFenetre, pour essai.

#### 2.6.6.1 D'abord par du code :

Dans HelloApplication.java ajouter le code suivant avant  
`stage.setScene(scene);`  
 :

```
HelloController controller = fxmlLoader.getController();
controller.btnFenetre.getStyleClass().setAll("btn","btn-danger");
scene.getStylesheets().add(BootstrapFX.bootstrapFXStylesheet());
```

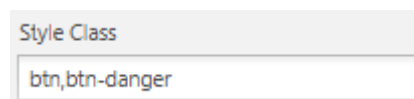
Essayer la vue.

#### 2.6.6.2 Ajouter du style depuis Scene Builder

Commenter la ligne

```
controller.btnFenetre.getStyleClass().setAll("btn","btn-danger");
```

Puis dans Scene Builder , pour le bouton Ouvrir, renseigner Style Class comme suit :

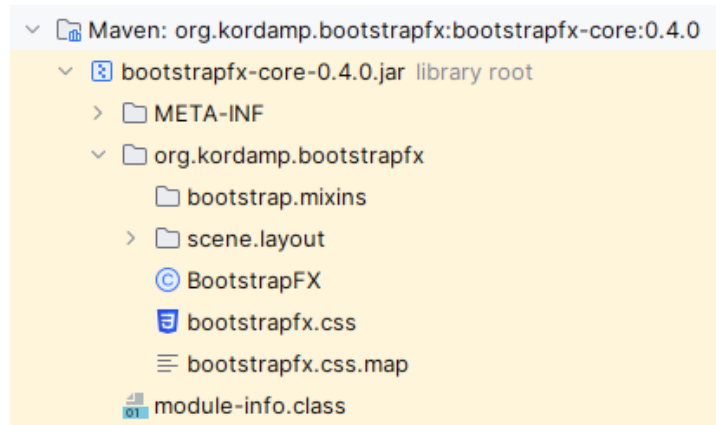


btn et btn-danger sont deux styles de BootstrapFX.

Le résultat n'est pas visible dans Scene Builder mais l'application fonctionne.

Pour que le résultat soit visible dans Scene Builder (je n'ai pas trouvé mieux) :

- Sous IntelliJ, dans external Libraries ouvrir

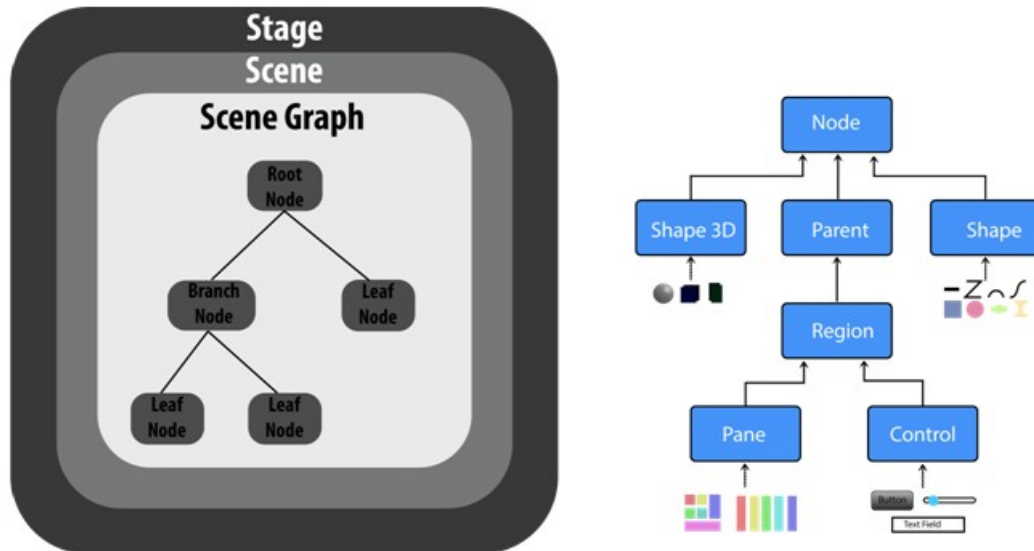


puis afficher bootstrapfx.css. Faire un Copier du contenu et le coller dans un fichier myBootstrapfx.css dans le répertoire contenant les .fxml

Sous Scene Builder, pour le VBox, renseigner la propriété Stylesheets avec ce fichier. Le bouton doit alors s'afficher en rouge ...

## Section 3 Retour sur la théorie

Rappel : Une vue dans Java FX est un contenu arborescent.



### 3.1 Les conteneurs de Controls : `javafx.scene.layout`

#### Gestionnaires de mise en page des noeuds

- **BorderPane** : top, bottom, right, left, or center region.
- **HBox** arranges nodes horizontally in a single row.
- **VBox** arranges nodes vertically in a single column.
- **StackPane** : nodes in a back-to-front single stack.
- **GridPane** : grid of rows and columns in which to lay out nodes.
- **FlowPane** : nodes in either a horizontal or vertical "flow," using specified boundaries.
- **TilePane** : nodes in uniformly sized layout cells or tiles
- **AnchorPane** : create anchor nodes to the top, bottom, left side, or center of the layout.

<https://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html> clic sur 1 Using Built-in Layout Panes

### 3.2 Les Controls JavaFX

Ce sont des éléments graphiques utilisables dans des conteneurs.



[https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui\\_controls.htm#JFXUI336](https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336)

### 3.3 Les Styles avec CSS

#### Principes

- La décoration d'une *scene* sera définie par un fichier CSS
- ⇒ enlève les lignes de mise en forme du code Java

#### Ajout d'un style à une *scene*

```
scene.getStylesheets()
.add(MyClass.class.getResource("Login.css"))
.toExternalForm();
```

#### 3.3.1 Définition de styles globaux pour la scene

##### Login.CSS affecte tous ces composants

```
.root {
  -fx-background-image: url("background.jpg");
}
.label {
  -fx-font-size: 12px;
  -fx-font-weight: bold;
  -fx-text-fill: #333333;
  -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
}
.button {
  -fx-text-fill: white;
  -fx-font-family: "Arial Narrow";
  -fx-font-weight: bold;
  -fx-background-color: linear-gradient(#61a2b1, #2A5058);
  -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );
}
.button:hover {
  -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
```

L'usage du '.' avant un composant est celui des CSS du Web. class=



### 3.3.2 Définition de styles pour des composants identifiés

Dans le monde du Web ce sont les tags avec id=

#### Login.java : définition des id des composants

```

    . . .
    scenetitle.setId("welcome-text");
    . . .
    actiontarget.setId("actiontarget");

```

#### Login.CSS styles des composants identifiés

```

/*id du composant*/
#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6, 0.0 , 0 , 2 );
}
/*id du composant*/
#actiontarget {
    -fx-fill: FIREBRICK;
    -fx-font-weight: bold;
    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
}

```

### 3.3.3 Faire connaître la feuille de style au fichier fxml

Il faut utiliser la balise <stylesheets>

#### fxml\_example.fxml : style / id / css loading

```

    . . .
    <GridPane fx:controller="fxml_example.FXMLExampleController"
        xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10"
        styleClass="root">
    . . .
    <Text id="welcome-text" text="Welcome"
        GridPane.columnIndex="0" GridPane.rowIndex="0"
        GridPane.columnSpan="2"/>
    . . .
    <stylesheets>
        <URL value="@Login.css" />
    </stylesheets>

    /*The @ symbol before the name of the style sheet in the URL
    indicates that the style sheet is in the same directory as the FXML file.*/

    </GridPane>

```

La balise <stylesheets> peut être mise en dehors de tout Control ou être mise dans le Control Layout de plus haut niveau – seul choix possible depuis Scene Builder.

## Section 4 Application avec du style ...

---

Nous allons coder la fenêtre de login proposée sur le site

<https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm>



### 4.1 Créer le projet

En utilisant IntelliJ :

Créer le projet en JavaFX de nom FXLogin

### 4.2 Faire connaître un fichier de style

Dans resources au même niveau que le fichier hello-view.fxml créer le fichier login.css

Y placer ce code

```
.root {
    -fx-background-image: url("background.jpg");
}
```

Récupérer l'image background.jpg et la placer au même endroit

Pour faire connaître le fichier login.css, ajouter dans la VBox de hello-view.fxml le code suivant

```
<stylesheet>
    <URL value="@login.css" />
</stylesheet>
```

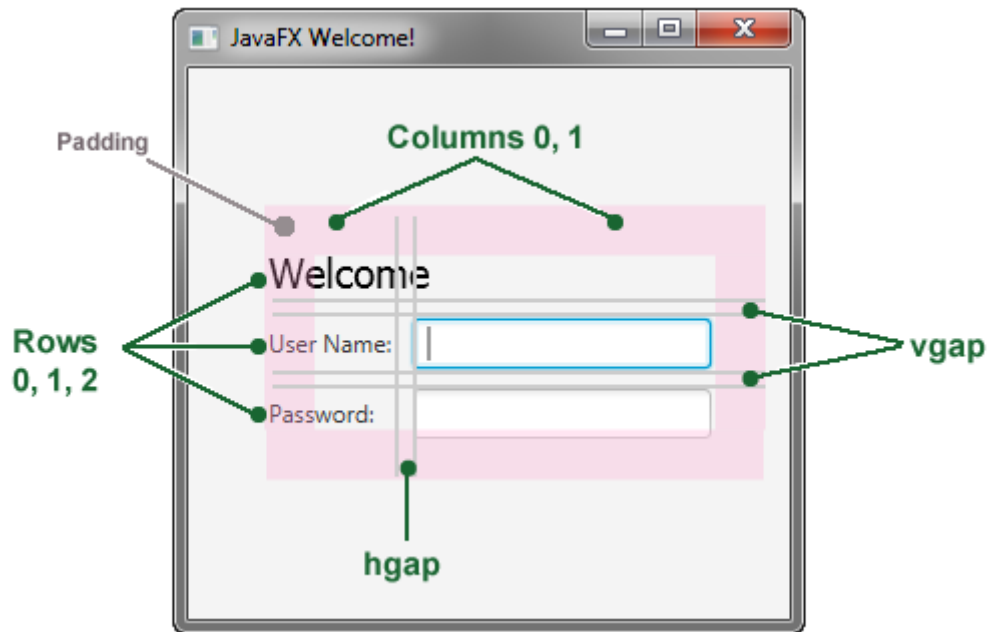
Renommer le fichier hello-view.fxml en login.fxml par un refactor

Renommer HelloController par LoginController

Renommer HelloApplication par LoginApplication

### 4.3 Placer les Layouts et Controls

Nous souhaitons réaliser ceci :



Le Layout principal est un GridPane.

Il a un alignement de type center

un Vgap et Hgap de 10 pixels

Les Controls utilisés sont :

- Un Label pour Welcome
- Un label pour User Name et Password
- Un TextField pour le nom
- un PasswordField pour le pswd

Le lien

[https://docs.oracle.com/javafx/scenebuilder/1/user\\_guide/library-panel.htm](https://docs.oracle.com/javafx/scenebuilder/1/user_guide/library-panel.htm)

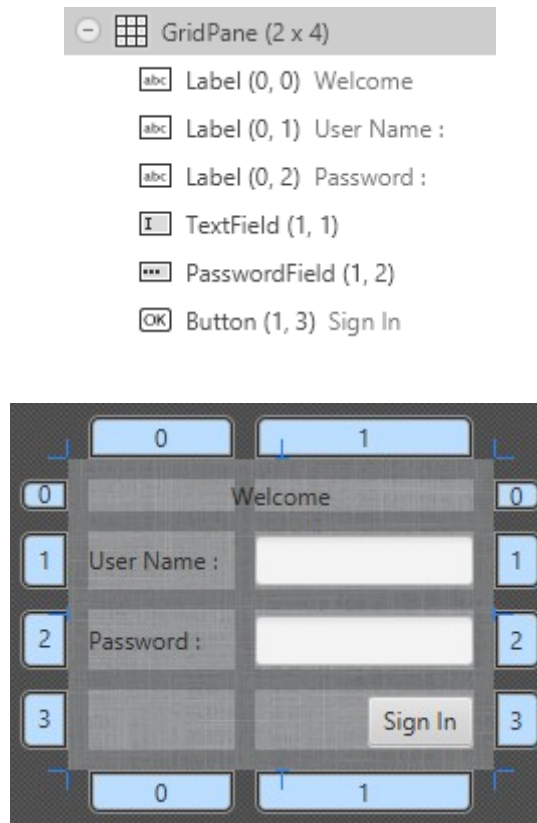
permet de comprendre comment ajouter des lignes et colonnes à un GridPane

Le lien

<https://openjfx.io/javadoc/22/javafx.graphics/javafx/scene/doc-files/cssref.html> fournit les attributs CSS disponibles

Pour ajouter des lignes et colonnes à un GridPane, ouvrir le menu contextuel sur les languettes numérotées.

Essayer d'obtenir ceci :



Dans login.css écrire une règle :

- pour la classe button pour une couleur de fond lightblue
- pour la classe text pour une taille de fonte de 20. Que ce passe t il si on met une règle sur la classe label plutôt que text ?

Tester.

## 4.4 Fenêtre de confirmation

Pour l'entraînement, une fenêtre modale s'ouvre pour indiquer le login correct avec le nom de la personne. Un bouton permet de fermer la fenêtre



Pour fermer la fenêtre ce code peut aider :

```
public void onClose(ActionEvent actionEvent) {  
    Stage stage = (Stage)lblResult.getScene().getWindow();  
    stage.close();  
}
```

## Section 5 Data Binding

---

Le sujet sur

[https://www.lirmm.fr/~pvalicov/Cours/ihm/Proprietes-Bindings\\_handout.pdf](https://www.lirmm.fr/~pvalicov/Cours/ihm/Proprietes-Bindings_handout.pdf)

<https://edencoding.com/javafx-properties-and-binding-a-complete-guide/>

En JavaFX un binding est un objet qui matérialise la liaison entre une valeur donnée et une ou plusieurs valeurs observables

Toutes les propriétés JavaFX supportent les bindings.

### 5.1 Binding unidirectionnel

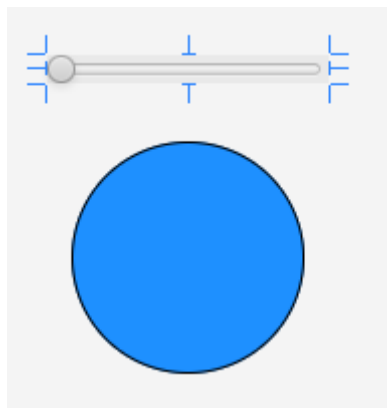
Dans l'application de login précédente nous allons créer une autre page dans laquelle un objet slider va changer le rayon d'un cercle.

Binding unidirectionnel : un changement du slider entraînera un changement du rayon du cercle (mais pas l'inverse).

Dans la fenêtre de login, ajouter un bouton nommé « Binding »

Il permet l'ouverture d'une nouvelle fenêtre nommée Binding.

Elle contient un Slider et un Cercle, comme ceci :



L'id du cercle est `cercle` et celui du slider `sldSlider`

Depuis le code `Slider.fxml` demander la création de la classe `Binding` et y placer les attributs `cercle` et `sldSlider`.

Depuis le code du bouton `Binding`, ajouter le code d'ouverture de la fenêtre.

Tester.

Nous n'allons pas créer un traitement d'événement mais faire un binding.

Dans Binding.java ajouter la méthode suivante :

```
public void setBinding() {
    Cercle.radiusProperty().bind(sldSlider.valueProperty());
}
```

Appeler cette méthode depuis le code du bouton Binding avec ce code :

```
Binding controller = fxmLoader.getController();
controller.setBinding();
```

Tester et constater que le slider agit sur le rayon du cercle

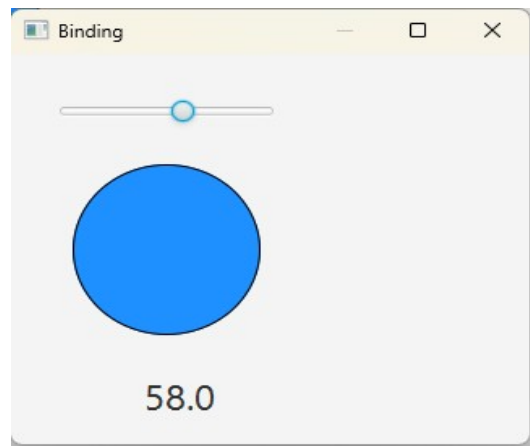
Le binding est donc réalisé par l'instruction :

```
Cercle.radiusProperty().bind(sldSlider.valueProperty());
```

La méthode bind() est à réaliser entre Propriétés.

Ajouter sous le cercle un label et faire en sorte qu'il affiche la valeur du slider.

Obtenir ceci :



Une propriété ne peut être liée de manière unidirectionnelle qu'à une seule autre propriété : un seul bind() possible.

On souhaite maintenant que la modification du rayon du cercle par la souris agisse sur le slider, et pas l'inverse

Sur le cercle utiliser l'événement Mouse Dragged pour y modifier son rayon.

Modifier le bind entre le cercle et le slider...

Tester

## 5.2 Binding bi-directionnel

Nous sommes prêts à utiliser le bind bi-directionnel.

Modifier le bind actuel entre le cercle et le slider par un bindBidirectional.

Tester.

Création des bindings haut niveau

La classe utilitaire Bindings possède des méthodes statiques utiles pour créer des liaisons haut niveau :

- calculs : add, divide, multiply, max, min, etc.
- logique : and, or, equal, lessThan, greaterThan, etc.
- conversion en chaînes de caractères : concat, convert, etc.
- expression ternaire :

when(condition).then(value1).otherwise(value2)

```
public void setBinding2(){
    cercle.fillProperty().bind(
        when(sldSlider.valueProperty().lessThan(20))
            .then(Color.WHITE)
            .otherwise(when(sldSlider.valueProperty().lessThan(40))
                .then(Color.GREEN)
                .otherwise(when(sldSlider.valueProperty().lessThan(60))
                    .then(Color.YELLOW)
                    .otherwise(when(sldSlider.valueProperty().lessThan(80))
                        .then(Color.RED)
                        .otherwise(Color.BLUE))));
    }
}
```

## 5.3 Placer un écouteur sur une propriété

L'interface Property implémente l'interface Observable

Il est donc possible d'ajouter des écouteurs sur une propriété

Sur élargissement de la fenêtre on souhaite redimensionner le cercle :

Dans le code de setBinding() ajouter ce code. Regarder et tester.

```
Scene scene = sldSlider.getScene();
scene.widthProperty().addListener(
    (source, ancienneValeur, nouvelleValeur) ->
        cercle.setRadius(nouvelleValeur.doubleValue()/4));
```



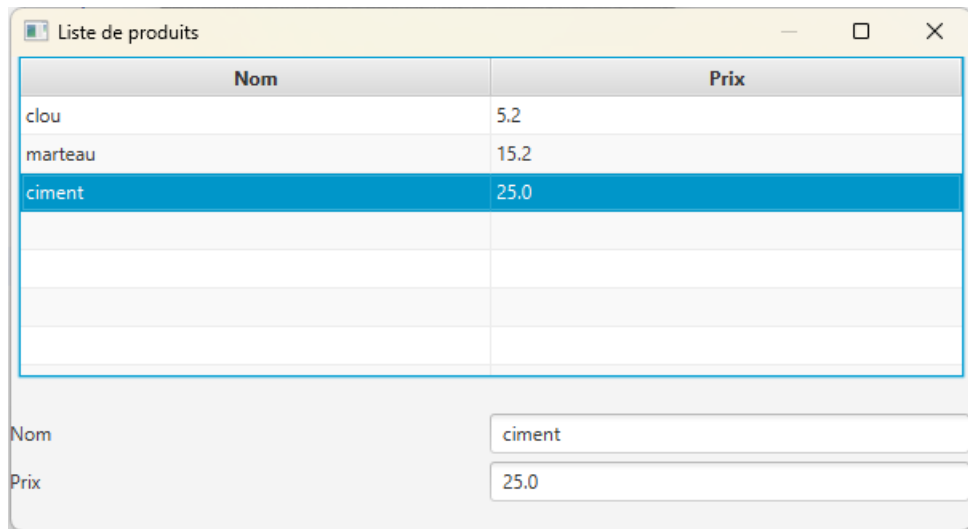
## Section 6 utiliser un TableView

---

### 6.1 But

Dans la fenêtre de login un bouton va être ajouté.

Il permet l'ouverture d'une fenêtre contenant un TableView qui affiche une liste de produits. La partie basse montre le détail du produit sélectionné.



### 6.2 Mode opératoire

Dans la fenêtre login ajouter un bouton nommé Liste.

Créer un package model et y placer une classe Product

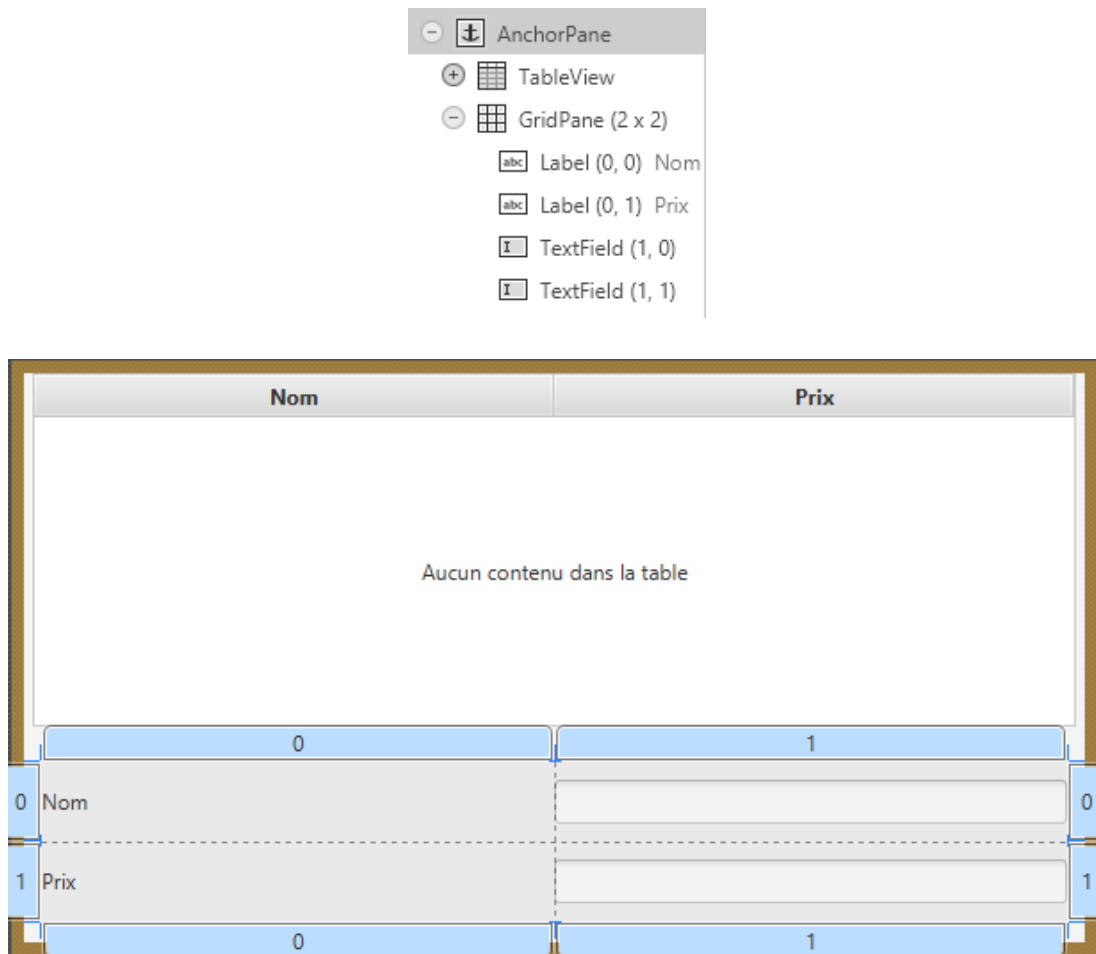
Mettre dans cette classe :

- name de type StringProperty
- price de type FloatProperty
- un constructeur qui a pour paramètres une String et un Float

Créer les accesseurs avec IntelliJ. Noter que des xxxProperty() vont aussi être créés. Il sont importants pour réaliser le binding.

Créer ensuite la nouvelle vue : ProductList.fxml

S'inspirer de ceci :



Les colonnes ont comme identifiant colName et colPrice.

Les textField ont comme identifiant txtName et txtPrice. Les événements onAction ont pour nom onNameChanged et onPriceChanged

La tableView a pour identifiant `tblvProductList`

Depuis le fichier ProductList.fxml demander à créer le fichier ProductListController et y placer les attributs qui correspondent aux identifiants ci-dessus.

Adapter le code du bouton Liste pour ouvrir la nouvelle fenêtre.

Tester.

### 6.2.1 Créer une liste de produits

Dans la classe ProductListController :

Modifier les attributs existants pour obtenir ceci :

```
@FXML
private TableView<Product> tblvProductList;
@FXML
private TableColumn<Product, String> colName;
@FXML
private TableColumn<Product, Float> colPrice;
```

Ajouter ce code pour obtenir une liste Observable par le binding

```
public ObservableList<Product> getProductList() {

    ObservableList<Product> products = FXCollections.observableArrayList();
    products.add(new Product("clou", 5.20f));
    products.add(new Product("marteau", 15.20f));
    products.add(new Product("ciment", 25f));

    return products;
}
```

### 6.2.2 Lier la liste à la TableView

Puis ajouter la méthode suivante qui est appelée automatiquement par le framework.

Elle fait le lien (binding) entre les **données** nameProperty et priceProperty et le **graphisme** des cellules des colonnes colName et colPrice

```
@FXML
public void initialize() {
    colName.setCellValueFactory(cellData -> cellData.getValue().nameProperty());
    colPrice.setCellValueFactory(cellData ->
cellData.getValue().priceProperty().asObject());
    tblvProductList.setItems(getProductList());
}
```

tester. La grille doit s'afficher remplie.

### 6.2.3 Affichage du détail d'un produit

Sur clic sur une ligne de la table on souhaite afficher le détail en dessous.

Ajouter sur la tableView l'événement « on Mouse Pressed » `onMousePressed`

Réaliser le code pour mettre à jour les 2 textField.

Tester et constater que le détail s'affiche.

### 6.2.4 Modifier la liste des produits

Depuis les 2 txtField, permettre qu'une saisie modifie les informations dans la liste.

La ligne

`tblvProductList.getSelectionModel().getSelectedItem()`

sera utile pour récupérer le produit sélectionné dans le TableView ...

## Section 7 La gestion des threads

---

La plupart des bibliothèques graphiques nécessitent à ce que le code graphique soit exécuté sur un même thread, sinon une exception est générée.

Une application graphique exécute une boucle infinie nécessaire pour capter les événements clavier et souris, tournant sur ce thread.

Java FX suit ce principe.

Nous allons l'expérimenter ici et apporter une solution.

### 7.1 Enoncé

On ajoute à l'application login un bouton nommé thread qui ouvre une fenêtre pendant 5 s et la ferme automatiquement.

Fonctionnellement ceci n'a aucun intérêt, c'est pour présenter les threads !

### 7.2 Travail à faire

Placer un `println` montrant le thread utilisé, à divers endroits du code existant.

Par exemple dans le `main`, le `start`, la fonction 'événement d'appel de la vue `ProductList` ...

```
System.out.println("onList " + Thread.currentThread());
```

Que constatez vous ?

Ajouter un bouton dans la fenêtre de login nommé Thread.

Créer une fenêtre `ThreadView.fxml`

Y placer seulement un Label avec un texte de votre choix.

Dans la fonction événement `onThread`, appeler la vue et utiliser un `show()` plutôt que `showAndWait`.

Assurez vous que l'on est pas bloqué dans le code.

Tester.

Ajouter ensuite une tempo de 5s avant de cacher cette fenêtre.

```
Thread.sleep(5000);
stage.hide();
```

Tester. Est ce que cela fonctionne ?

Il faut lancer cette tempo dans un autre Thread.

Exemple de code qui remplace les 2 lignes suivantes :

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            Thread.sleep(5000);
            System.out.println("sleep " + Thread.currentThread());
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}).start();
stage.hide();
```

Tester. Est ce que cela fonctionne ?

Placer alors stage.hide() dans le thread de la tempo.

Tester. Est ce que cela fonctionne ? Pourquoi ?

Il faut placer l'exécution du hide() dans le JavaFX Application Thread.

L'instruction Platform.runLater() permet cela :

```
Platform.runLater(new Runnable() {
    @Override
    public void run() {
        System.out.println("runLater" + Thread.currentThread());
        stage.hide();
    }
});
```

Tester

Ce code peut être plus court en utilisant les expressions lambda :

```
new Thread()->{
    try {
        Thread.sleep(5000);
        Platform.runLater()->{
            stage.hide();
        });
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
}).start();
```

### 7.3 Résumé

Toutes les instructions concernant du graphisme ou binding vers du graphisme doivent être exécutée par le JavaFX Application Thread.

Tout traitement asynchrone – recherche de données distantes, temporisations ... - doivent se faire dans d'autres threads pour ne pas geler momentanément l'interface graphique (la fenêtre devient blanche, un sablier apparaît ...).

Le retour d'un autre thread vers le monde graphique se fait par Platform.runLater()