



BTS-SIO

Conception et développement d'applications

Atelier JavaFx

ATELIER N° 1



Objectifs

Le développement informatique concerne tout ce qui touche à l'étude, à la conception, à la construction, au développement, à la mise au point, à la maintenance et à l'amélioration des logiciels et autres applications et sites web. C'est le développeur informatique qui en a la charge. Il a plusieurs rôles : analyser les besoins des clients/utilisateurs, s'occuper de l'écriture informatique, rédiger les notices...etc.

Il doit posséder de multiples compétences : connaissance du langage informatique, expertise des technologies de bases de données.

JavaFX est un Framework et une bibliothèque d'interface utilisateur issue du projet OpenJFX, qui permet aux développeurs Java de créer une interface graphique pour des applications de bureau, des applications internet riches et des applications smartphones et tablettes tactiles.

Cet atelier va nous permettre de réviser les bases de JavaFx.



Contrainte de cet atelier

Connaitre correctement les bases du langage Java et la POO.

Avoir déjà vu les bases du Framework **JavaFx**

Considérations techniques & logicielles

Eclipse IDE pour les phases de programmation.

JavaFx installé et configuré sous Eclipse



Bloc

Bloc de compétences n°2 : option B « Solutions logicielles et applications métiers » - Conception et développement d'applications



Titre

Atelier 01 JavaFX

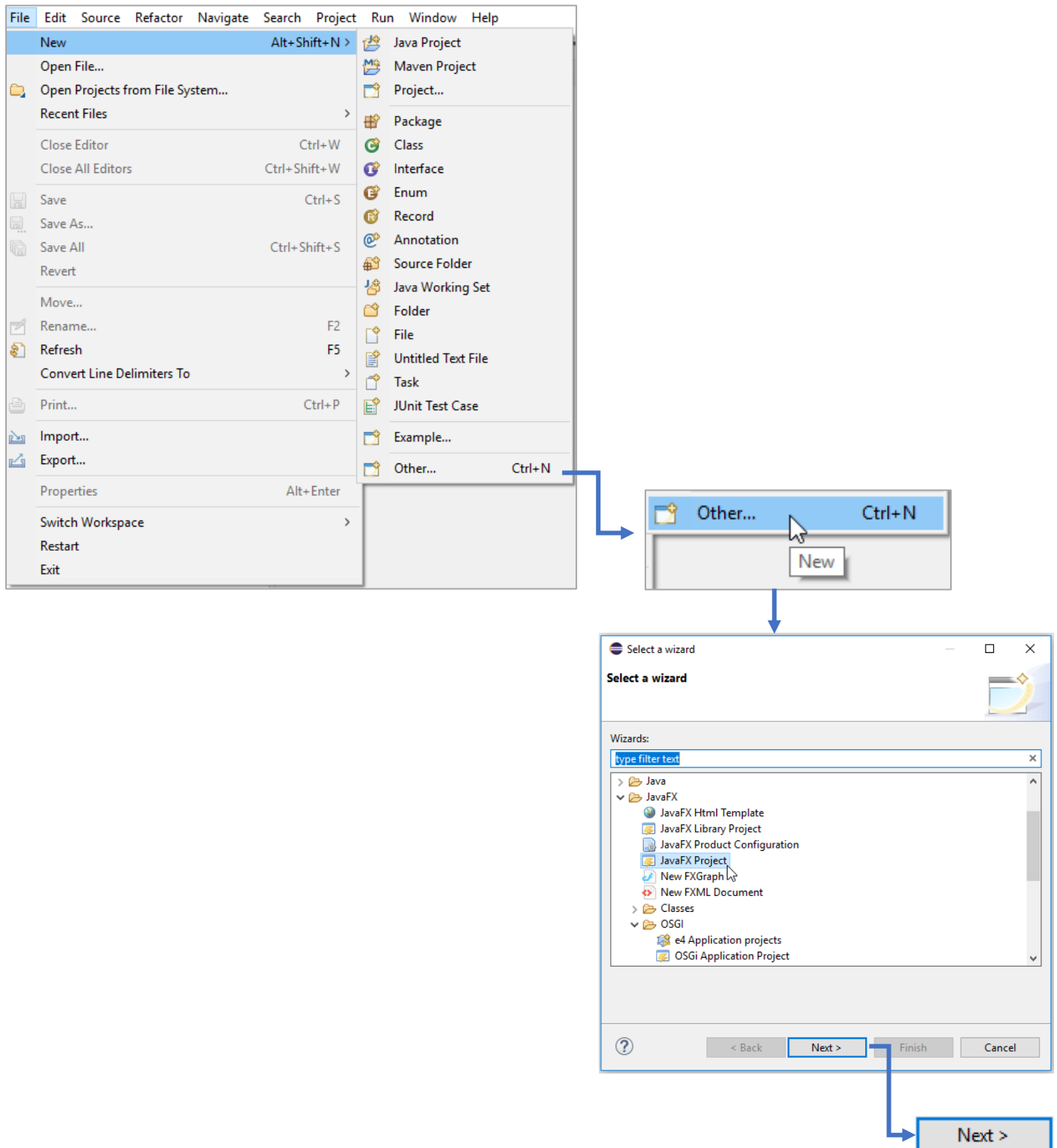
Table des matières

1. Projet.....	3
1.1. Création du projet.....	3
1.2. Première modification	7
1.3. Modification du fichier FXML.....	8
1.4. Déclaration des contrôles et évènements dans le contrôleur	10
1.5. Gestion des évènements.....	11
1.6. Test	12
2. Rajout d'une nouvelle fenêtre	12
2.1. Création des nouveaux objets	12
2.2. Traitement du bouton Annuler	15
2.3. Appel Modal.....	16
3. Passage de paramètres.....	18
3.1. Passage entre la première et la seconde fenêtre.....	18
3.2. Passage entre la seconde et la première fenêtre	19
3.3. Dernière amélioration, la gestion du bouton Annuler	21

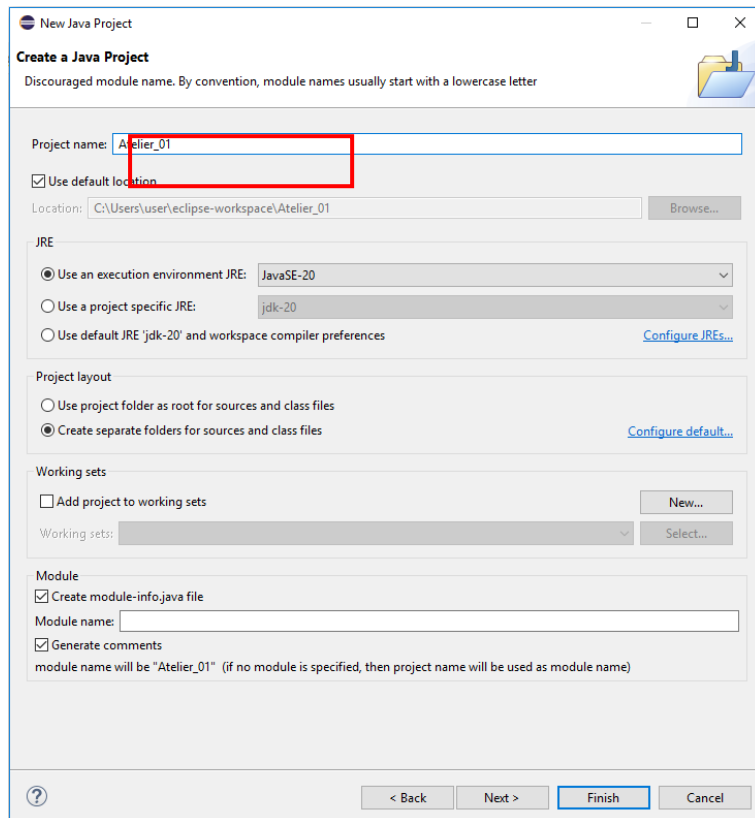
1. Projet

1.1. Création du projet

- Créez un nouveau projet JavaFx.

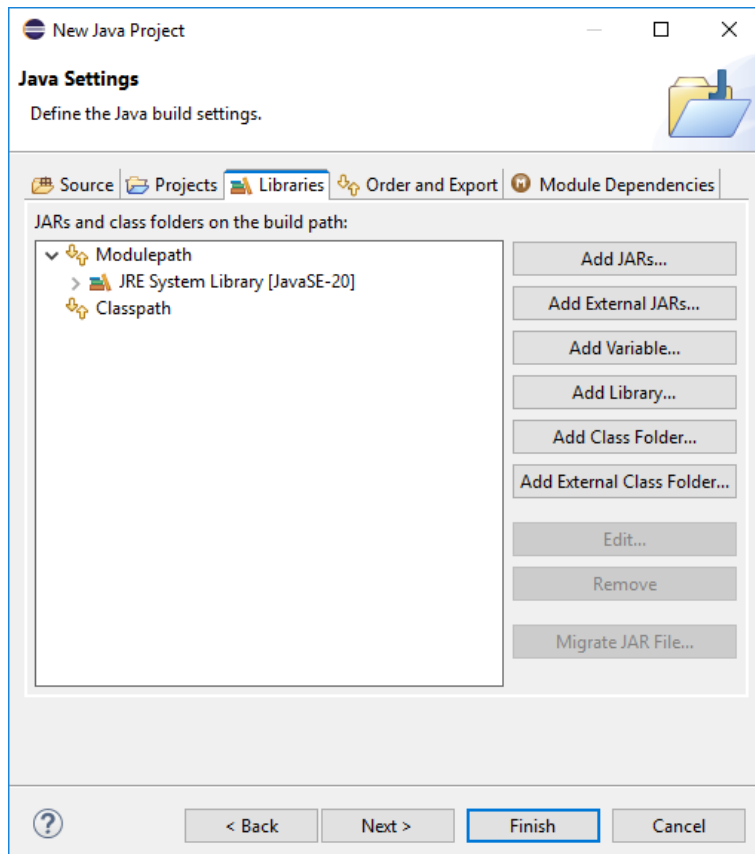


Donnez-lui comme nom **[Atelier-01]**



Cliquez sur Next

Next >



Venir sur l'onglet **Libraries**

Libraries

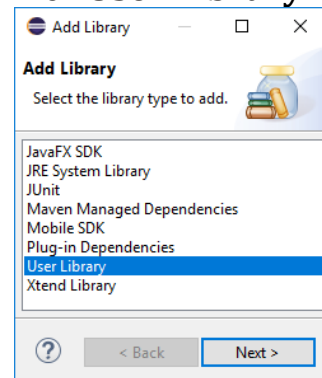
Cliquez sur la ligne **Modulepath**

Modulepath

Cliquez ensuite sur **Add Library...**

Add Library...

Sélectionnez **User Library**



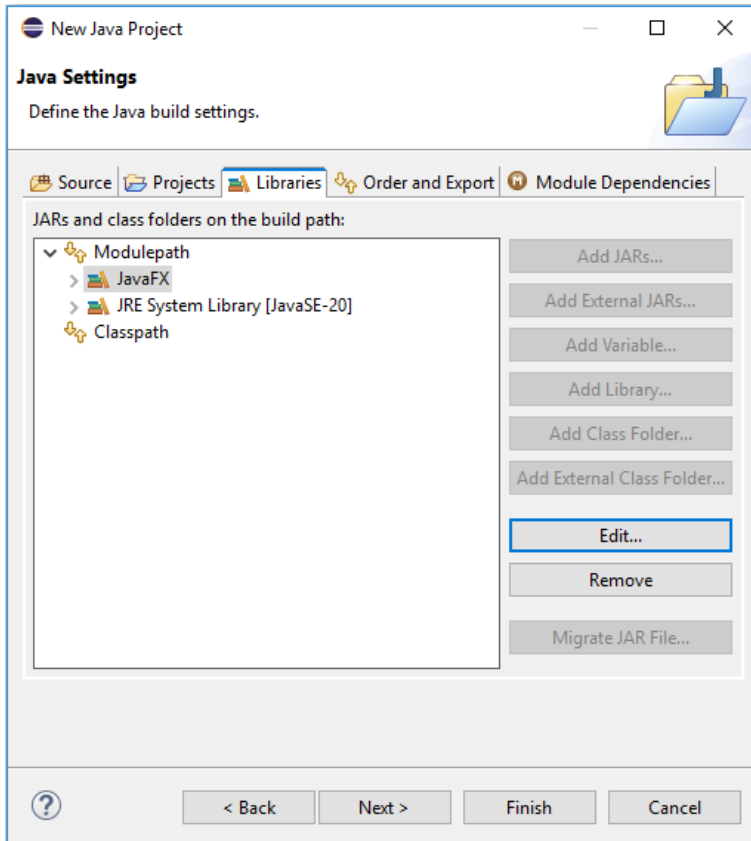
Cochez **Javafx**

User libraries:

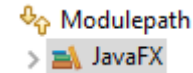
☒ JavaFX

puis cliquez sur **Finish**

Finish

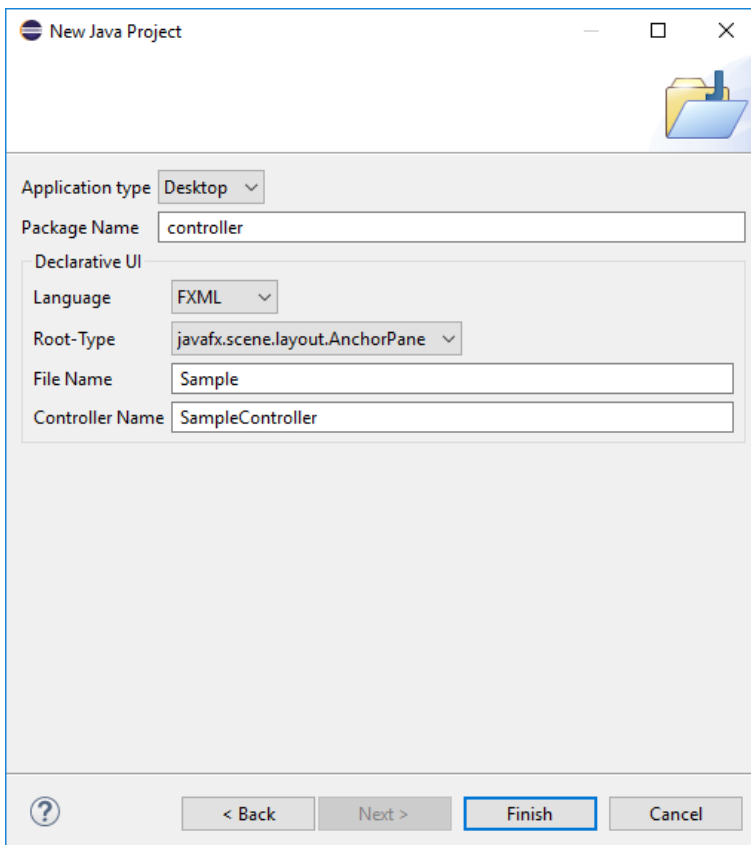


La ligne **JavaFx** doit apparaître dans le **Modulepath**

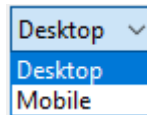


Cliquez sur Next

Next >

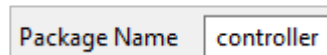


Le type d'application permet de définir la plateforme sur laquelle va être utilisée l'application. Il y a deux choix :

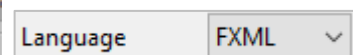
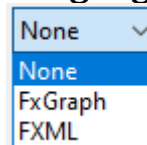


Nous allons utiliser le choix par défaut. Application de type bureau.

Saisir **controller** dans la zone **Package Name**



Choisir **FXML** dans la zone **Language**



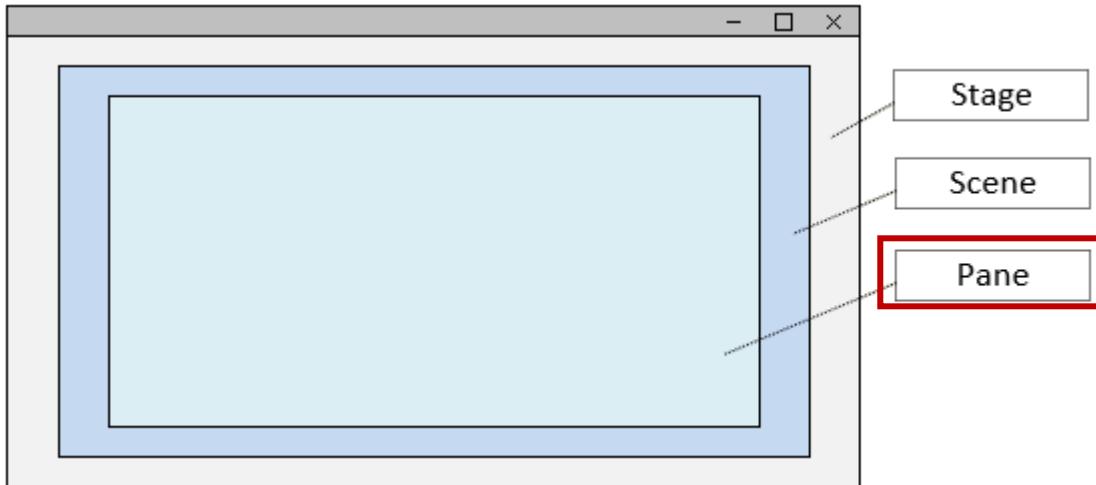
Choisir

javafx.scene.layout.AnchorPane

Dans la zone **Root-Type**

Root-Type javafx.scene.layout.AnchorPane ▾

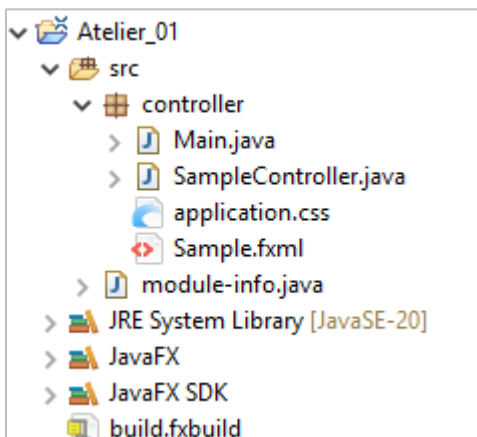
Il s'agit du type de la racine de notre fenêtre



Cliquez sur Finish

Finish

Vous devez voir dans le **Package Explorer**



1.2. Première modification

Si vous ouvrez la classe **Main.java** vous devez trouver le code suivant :



```
package controller;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.fxml.FXMLLoader;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            AnchorPane root =
(AnchorPane)FXMLLoader.Load(getClass().getResource("Sample.fxml
"));

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

            scene.getStylesheets().add(getClass().getResource("applica
tion.css")).toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Launch(args);
    }
}
```



Si vous avez des questions ou des doutes sur les lignes de code de cette classe, reportez-vous au document **FL-Fiche du langage javaFx.pdf**

Nous allons décomposer la ligne



```
AnchorPane root =
(AnchorPane)FXMLLoader.Load(getClass().getResource("Sample.fxml"));
```


Cette ligne fait l'intégralité du traitement de contrôle et de génération de la fenêtre en une seule ligne. Vous allez la remplacer par :



```
FXMLLoader loader = new FXMLLoader();  
loader.setLocation(getClass().getResource("Sample.fxml"));  
AnchorPane root = (AnchorPane) loader.load();
```

Vous allez également modifier la ligne suivante :



```
Scene scene = new Scene(root, 400, 400);
```

En supprimant les paramètres [400,400]

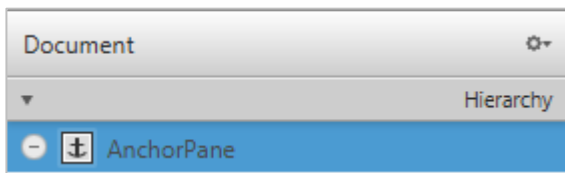
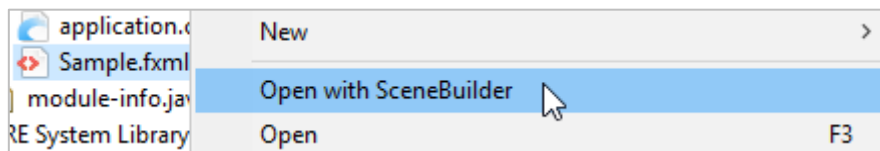
Rajoutez ensuite un titre à notre fenêtre en insérant la ligne suivante juste après `primaryStage.setScene(scene);`



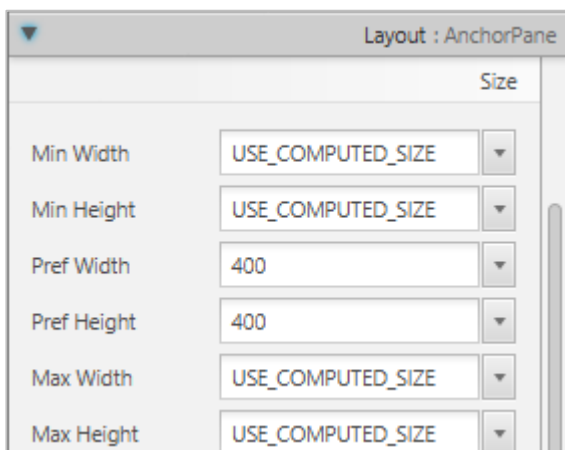
```
primaryStage.setTitle("Notre première fenêtre");
```

1.3. Modification du fichier FXML

Ouvrez le fichier [Sample.fxml] avec SceneBuilder



Cliquez à gauche sur **AnchorPane** (notre racine)



Dans le menu **Layout** à droite, saisir

- 400 dans **Pref Width**
- 400 dans **Pref Height**

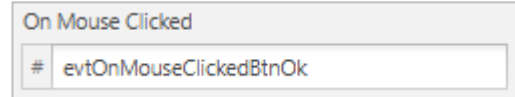
Et rajoutez les contrôles suivants [menu **Controls**]:

- Un contrôle TextField : TextField **Fx:id** → **txfText**
- Un contrôle Label : Label **Fx:id** → **lblText**

Premier bouton :

Fx:id → btnOk

Evènement

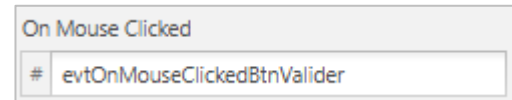


- Deux contrôles Button : 

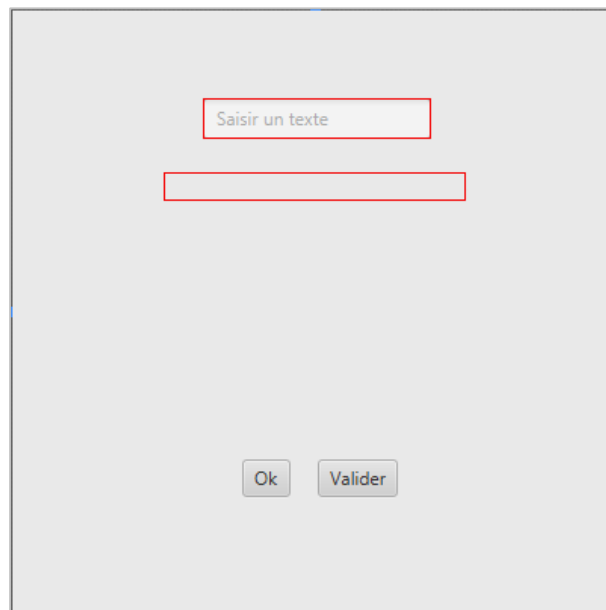
Second bouton :

Fx:id → btnValider

Evènement



L'ordre d'affichage est le suivant :



Sauvegardez et contrôler le fichier [Sample.fxml]

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="400.0" prefWidth="400.0" xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/20.0.1" fx:controller="controller.SampleController">
  <children>
    <TextField fx:id="txfText" layoutX="119.0" layoutY="58.0" promptText="Saisir un texte" />
    <Button fx:id="btnOk" layoutX="174.0" layoutY="157.0" mnemonicParsing="false"
onMouseClicked="#evtOnMouseClickedBtnOk" text="Ok" />
    <Label fx:id="LblText" layoutX="106.0" layoutY="257.0" prefWidth="200.0" />
    <Button fx:id="btnValider" layoutX="168.0" layoutY="194.0" mnemonicParsing="false"
onMouseClicked="#evtOnMouseClickedBtnValider" text="Valider" />
  </children>
</AnchorPane>
```




1.4. Déclaration des contrôles et évènements dans le contrôleur

Vous devez voir que certaines lignes ont des  ou des 

- Les  correspondant à des contrôles déclarés dans le fichier FXML et pas dans le contrôleur.

```
fx:id="txfText" layoutX=
```

The controller 'SampleController' has no field 'txfText'
Press 'F2' for focus

- Les  eux, correspondant à des évènements déclarés dans le fichier FXML mais dont la méthode est manquante dans le contrôleur.

```
onMouseClicked="#evtOnMouseClickedBtnValider"
```

The controller 'SampleController' has no event slot 'evtOnMouseClickedBtnValider'
Press 'F2' for focus

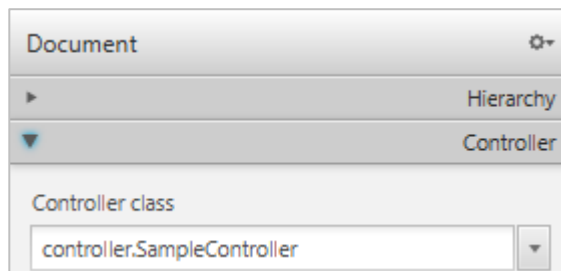
Ouvrez votre contrôleur [SampleController.java]



Le contrôleur d'un fichier FXML est défini sur la racine du fichier :

```
fx:controller="controller.SampleController"
```

vous pouvez également le voir dans SceneBuilder



Pour l'instant il n'y a pas grand-chose dedans



```
package controller;
```

```
public class SampleController {  
  
}
```

Les différents objets rattachés avec le fichier FXML doivent être déclarés avec l'annotation `@FXML` et avec la visibilité **private**. Le nom doit correspondre au paramètre **fx:id** du fichier **FXML**. La liaison se fait entre :

Fichier FXML

```
fx:id="txfText"
```

Et

Contrôleur Java

```
@FXML private TextField txfText;
```

Rajoutez les lignes suivantes :




```
package controller;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;

public class SampleController {
    /** Contrôles de la fenêtre */
    @FXML private TextField txfText;
    @FXML private Label lblText;
    @FXML private Button btnOk;
    @FXML private Button btnValider;
    /** Méthodes évènements JavaFx */
    @FXML private void evtOnMouseClickedBtnOk(){
    }
    @FXML private void evtOnMouseClickedBtnValider(){
    }
}
```



Regardez votre fichier FXML, vous allez voir que les  et les  ont disparus.

1.5. Gestion des évènements

Le programme doit permettre les traitements suivants :

- Un clic sur le bouton [Ok] affiche le contenu du **TextField** dans le **Label**,
- Un clic sur le bouton [Valider] va nous permettre d'afficher une nouvelle (*qui n'existe pas pour l'instant*). Nous ne ferons rien pour l'instant.

Dans notre contrôleur, rajouter le code suivant dans la méthode

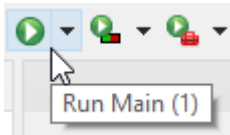
[evtOnMouseClickedBtnOk]



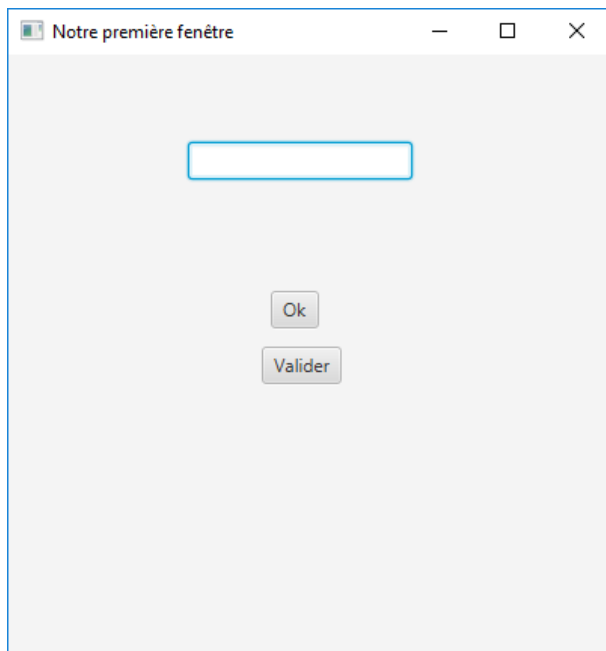
```
@FXML private void evtOnMouseClickedBtnOk(){
    /** Transfert de la valeur saisie du
     * TextField [txfText] vers
     * le Label [lblText]
     */
    lblText.setText(txfText.getText());
}
```

1.6. Test

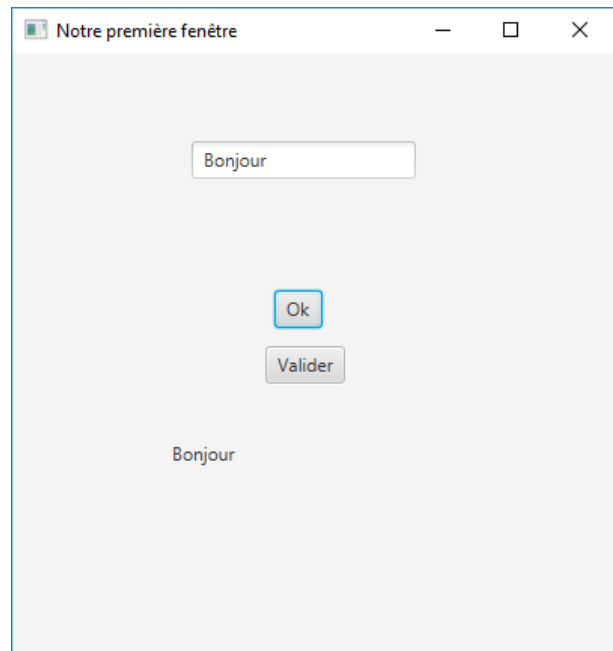
Lancez le programme :



Lancez l'application



Saisissez un texte dans la zone de saisie et cliquez sur le bouton [Ok]
Le texte doit apparaître dans le Label.



2. Rajout d'une nouvelle fenêtre

2.1. Création des nouveaux objets

Nous allons rajouter une nouvelle fenêtre dans l'application.

Créez un nouveau fichier **FXML** que nous allons appeler **Sample2.fxml**.

Ce fichier a :

- Comme contrôleur : le fichier **SampleContrôleur2.java**
- Trois contrôles



JavaFX - **textField**

JavaFX - **Button**

JavaFX - **Button**

fx:id → txfText

fx:id → btnValider

évènement :

evtOnMouseClickedBtnValider

fx:id → btnAnnuler

évènement :

evtOnMouseClickedBtnAnnuler

Vous devriez avoir dans votre fichier FXML :



```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="400.0" prefWidth="400.0" xmlns="http://javafx.com/javafx/20.0.1"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="controller.SampleController2">
<children>
    <TextField fx:id="txfText" layoutX="119.0" layoutY="58.0" promptText="Saisir un texte" />
    <Button fx:id="btnAnnuler" layoutX="128.0" layoutY="130.0" mnemonicParsing="false"
onMouseClicked="#evtOnMouseClickedBtnAnnuler" text="Annuler" />
    <Button fx:id="btnValider" layoutX="200.0" layoutY="130.0" mnemonicParsing="false"
onMouseClicked="#evtOnMouseClickedBtnValider" text="Valider" />
</children>
</AnchorPane>
```

Et dans votre contrôleur :



```
package controller;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class SampleController2 {
    /** Contrôles de la fenêtre */
    @FXML private TextField txfText;
    @FXML private Button btnAnnuler;
    @FXML private Button btnValider;
    /** Méthodes évènements JavaFx */
    @FXML private void evtOnMouseClickedBtnAnnuler(){
    }
    @FXML private void evtOnMouseClickedBtnValider(){
    }
}
```

En prenant exemple sur la première partie, nous allons, dans la méthode [evtOnMouseClickedBtnValider()] du contrôleur [SampleController.java] rajouter l'appel du fichier [Sample2.fxml]. Pour cela il suffit de copier le code qui se trouve dans la classe [Main.java] puis de le modifier :



```
@FXML private void evtOnMouseClickedBtnValider(){
    try {
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("Sample2.fxml"));
        AnchorPane root = (AnchorPane) loader.load();

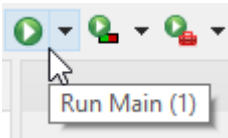
        Scene scene = new Scene(root);
        scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
        primaryStage.setScene(scene);
        primaryStage.setTitle("Notre seconde fenêtre");
        primaryStage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Les trois lignes primaryStage.**** sont en erreur, c'est tout à fait logique, primaryStage n'a pas été déclarée. Nous sommes en POO, il nous faut donc instancier cet objet.

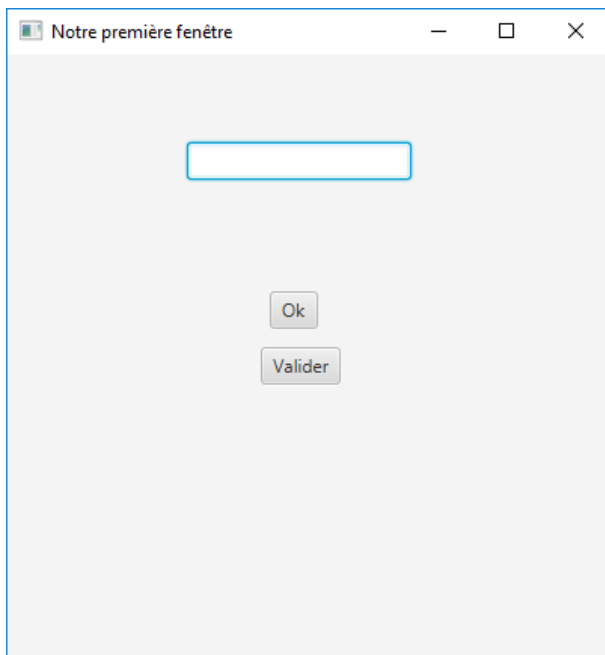


```
Stage primaryStage = new Stage();
primaryStage.setScene(scene);
```

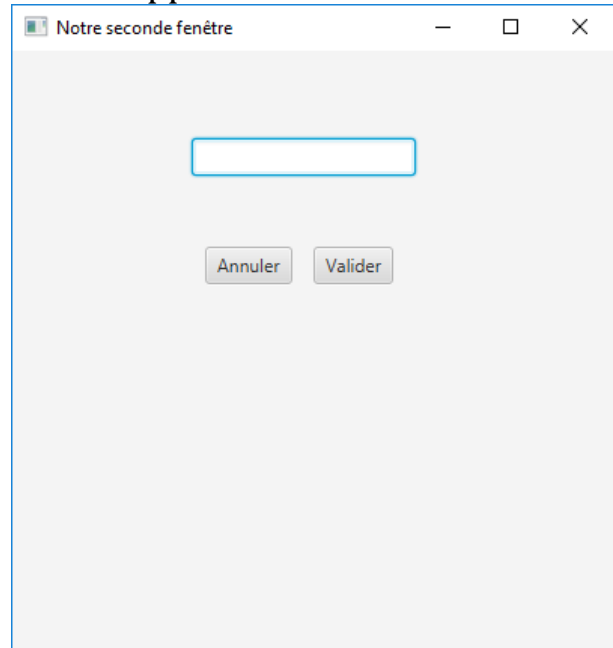
Vous pouvez à présent tester notre application :



Lancez l'application



cliquez sur le bouton **[Valider]**, la seconde fenêtre doit apparaître.



Par contre les événements liés aux boutons **[Annuler]** et **[Valider]** ne produisent rien. Ce qui est normal, il n'y a pas de code.

2.2. Traitement du bouton Annuler

Occupons-nous du bouton **[Annuler]**. Un clic dessus doit fermer la fenêtre et permettre de revenir sur la fenêtre appelante.

En **JavaFX**, pour fermer une fenêtre, il faut fermer l'objet **Stage** avec la méthode **close()**. Nous n'avons pas d'objet déclaré correspondant à cela. Pas de problème, nous allons l'instancier.

Ok mais à partir de quoi ?

Simplement en récupérant le Stage sur lequel est positionné le bouton **[Annuler]** par exemple.

Rajouter le code suivant dans la méthode **evtOnMouseClickedBtnAnnuler()** :

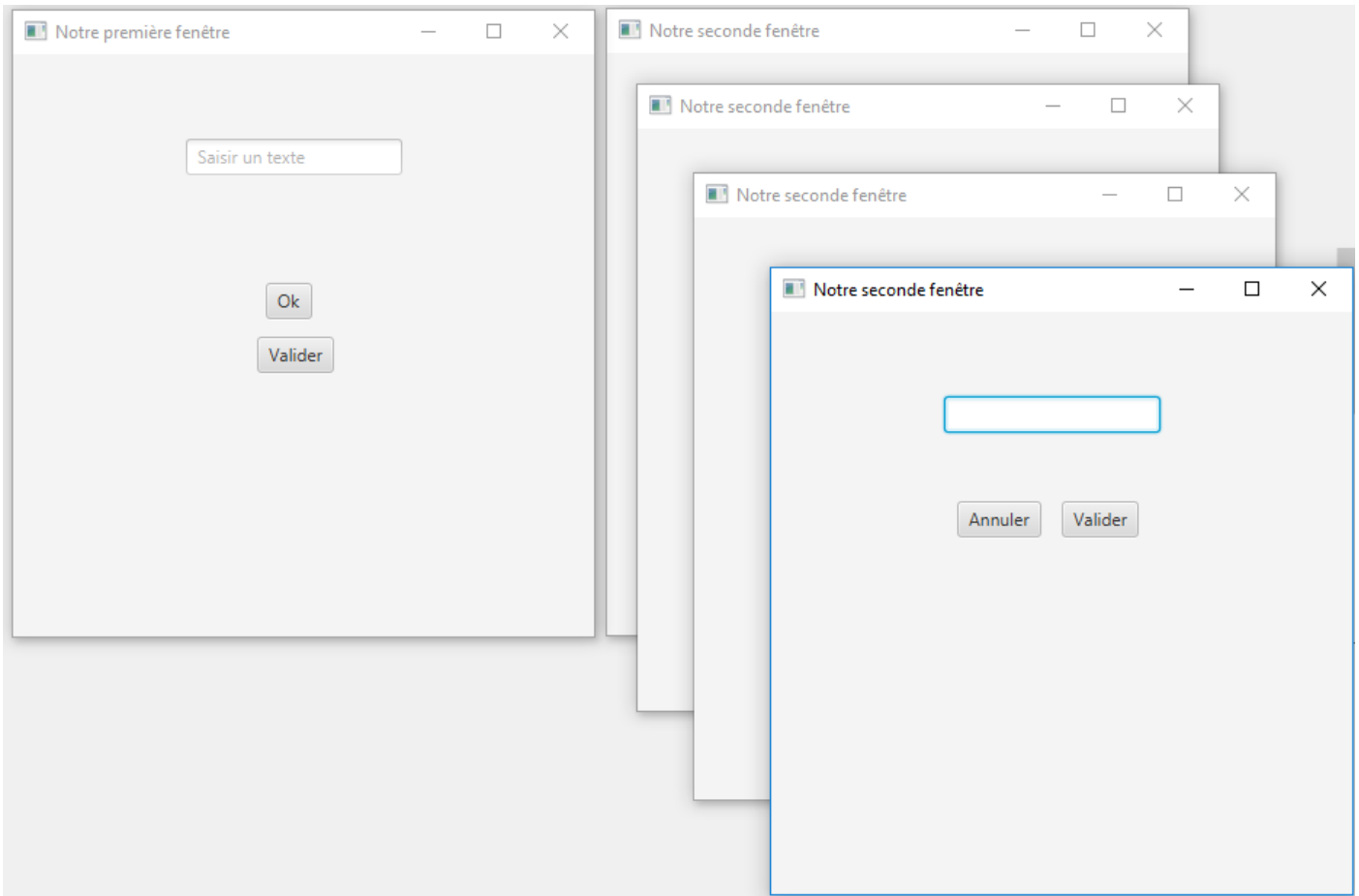


```
@FXML private void evtOnMouseClickedBtnAnnuler(){
    /** Récupération de l'objet Stage sur lequel
     * se trouve le bouton Annuler */
    Stage stage = (Stage) btnAnnuler.getScene().getWindow();
    /** Fermeture de la fenêtre */
    stage.close();
}
```

Refaites un test, cela marche.

2.3. Appel Modal

Néanmoins nous avons un petit problème, faites le test, relancez l'application et cliquez X fois sur le bouton [Valider] de la première fenêtre. Vous allez pouvoir ouvrir pleins de seconde fenêtre. Ce n'est pas très logique comme fonctionnement.



Il existe plusieurs manières d'afficher une fenêtre dans JavaFx en utilisant la propriété **[initModality]** de l'objet **[Stage]**.

- **APPLICATION_MODAL**
Permet de définir une fenêtre modale qui interdit tous les événements des autres fenêtres de l'application.
- **NONE**
Permet de définir une fenêtre qui n'est pas modale et ne bloque aucune autre fenêtre. (mode par défaut)
- **WINDOW_MODAL**
Permet de définir une fenêtre modale moins restrictive qu'APPLICATION_MODAL car elle interdit uniquement les événements des fenêtres mères.

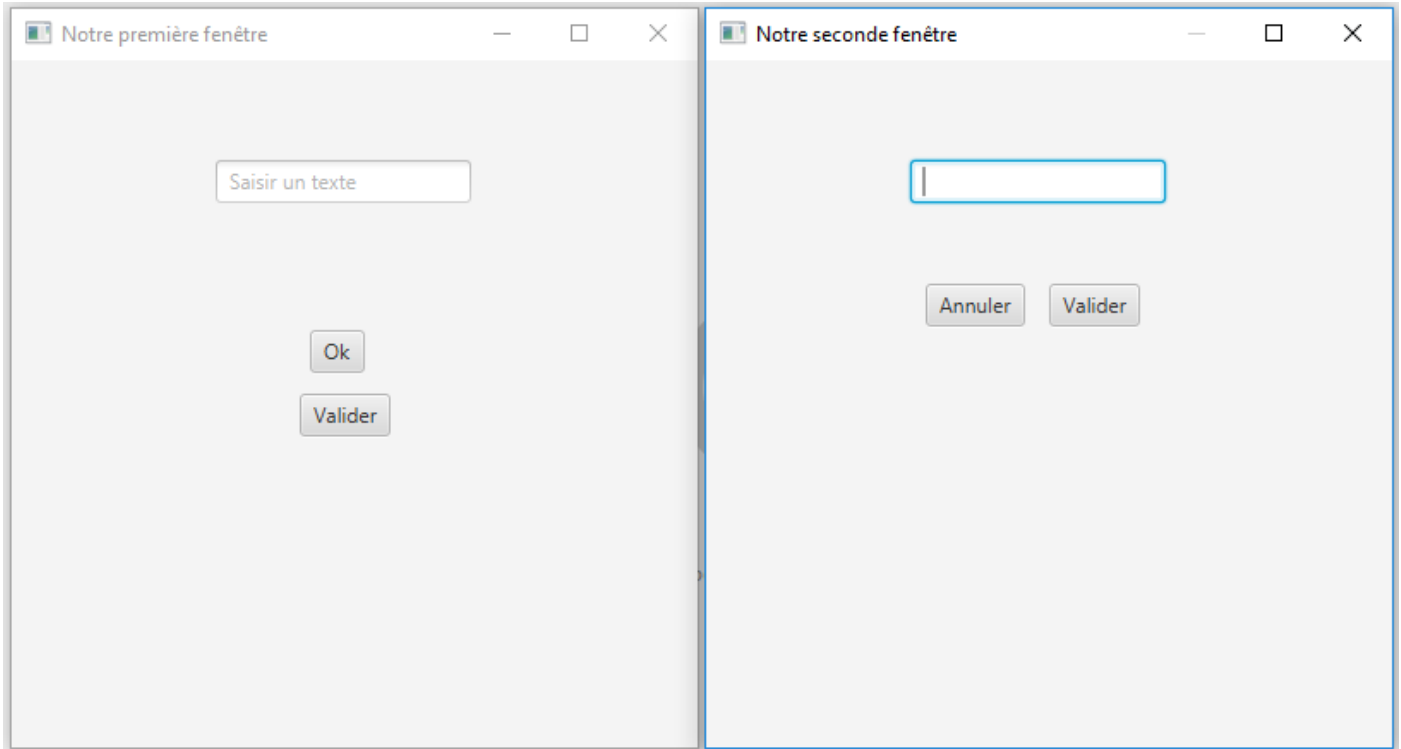
Nous utiliserons le premier choix **[APPLICATION_MODAL]**, il ne sera donc pas possible d'effectuer un traitement sur la première fenêtre de bord tant que la seconde sera ouverte.

Rajouter le code suivant dans la méthode **[evtOnMouseClickedBtnValider()]** du contrôleur **[SampleController.java]** juste avant l'appel de la méthode **[show()]**



```
primaryStage.initModality(Modality.APPLICATION_MODAL);
```

Refaites un test, vous allez voir que la première fenêtre est inaccessible tant que la seconde est « **ouverte** ».



Un clic sur le bouton [**Annuler**] ferme la seconde fenêtre et «**libère**» la première.

3. Passage de paramètres

3.1. Passage entre la première et la seconde fenêtre

Nous allons partir du principe que le bouton [**Valider**] appelle la seconde fenêtre mais lui passe également le contenu de la zone de saisie. Le passage de paramètres entre deux fenêtres javaFX est un traitement des plus classiques.

Pour cela nous allons devoir instancier le contrôleur de la seconde fenêtre dans la méthode [**evtOnMouseClickedBtnValider()**] de la première.

Rajoutez le code suivant dans cette méthode juste après l'instanciation du **Stage**.



```
SampleController2 controller = loader.getController();
```

Cette ligne crée un objet **controller** qui est de type **SampleController2** (*c'est une classe Java*). L'appel de la méthode **getController()** de l'objet **loader** permet de vérifier qu'il s'agit bien du contrôleur du fichier FXML chargé dans le **loader**.

A partir de ce moment, il est possible d'accéder à tous les attributs ou méthodes public du contrôleur [**SampleController2.java**].

Dans ce contrôleur, nous allons rajouter une méthode qui va mettre à jour la valeur du **TextField** [**txfText**] avec le paramètre que nous allons lui passer. Cette logique correspond à un Setter.

Rajouter la méthode suivante dans le contrôleur [**SampleController2.java**].



```
public void setTxfText(String txt) {  
    txfText.setText(txt);  
}
```

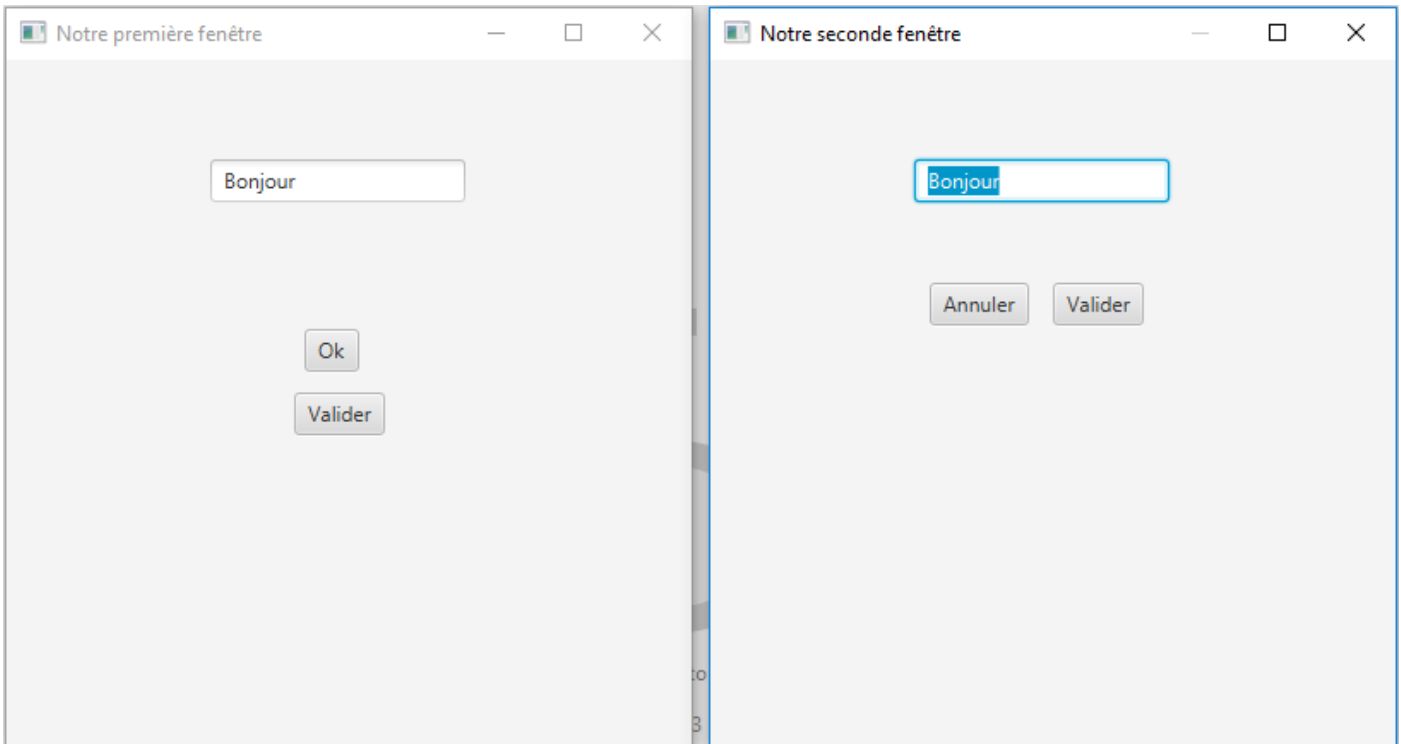
Cette méthode reçoit une chaîne de caractères et initialise le **TextField** [**txfText**] avec. Dans la méthode [**evtOnMouseClickedBtnValider()**] du contrôleur [**SampleController.java**] juste après l'instanciation du **controller** (*voir ci-dessus*) :



```
controller.setTxfText(txfText.getText());
```

Cette ligne appelle la méthode **public setTxfText** du **controller** et lui passe le contenu de la **Textfield** [**txfText**].

Faites un test :



3.2. Passage entre la seconde et la première fenêtre

Dans de nombreux cas, le programme appelé peut renvoyer une valeur au programme appelant. C'est ce que nous allons faire. Notre seconde fenêtre va renvoyer la valeur saisie dans le TextField [**txfText**] à la première.

La logique est exactement la même, cette fois-ci notre contrôleur

[**SampleController2.java**] va retourner la valeur du TextField. Cela correspond à la logique d'un Getter.

Dans ce même contrôleur, rajouter un Getter sur le TextField [**txfText**].



```
public String getTxfText() {  
    return txfText.getText();  
}
```

Actuellement, le contrôleur [**SampleController.java**] dans la méthode [**evtOnMouseClickedBtnValider()**] affiche la seconde fenêtre en faisant un **primaryStage.show()**.

Pour pouvoir récupérer la valeur saisie dans la seconde fenêtre, il faut rajouter du code après cette instruction (*au retour, après la fermeture de la seconde fenêtre*). La méthode **.show()** ne gère pas ce cas de figure, elle s'exécute et ne prend pas en compte les lignes se trouvant après.

Nous allons changer cette méthode en **showAndWait()**. Là le programme se met en attente de la fermeture de la fenêtre appelante puis reprend le code en séquentiel.

Derrière il suffit d'appeler le Getter pour récupérer le texte saisi puis de l'insérer dans le TextField de la première fenêtre.

Modifiez votre code comme ci-dessous :



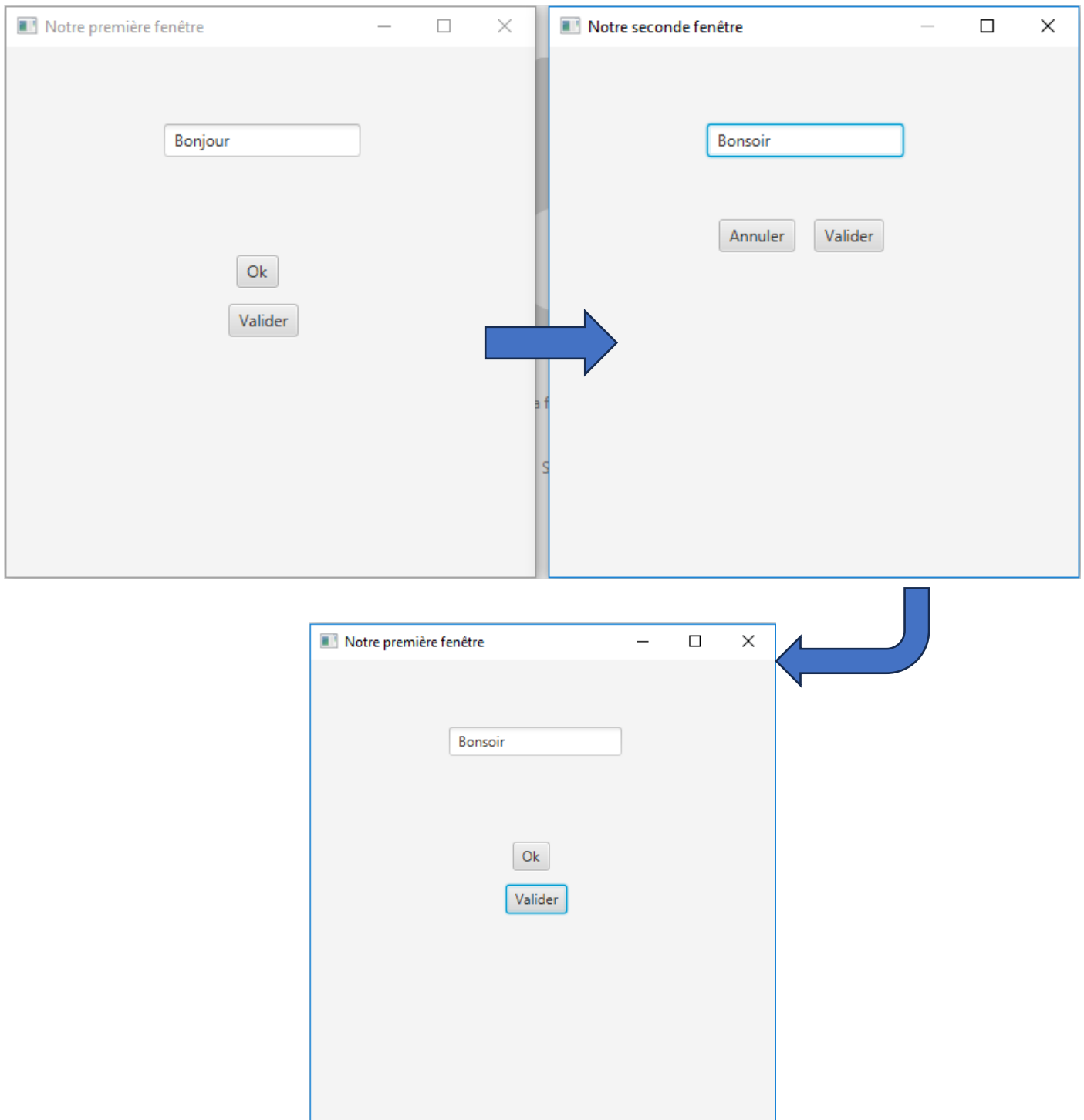
```
/** Suppression de la méthode show()  
 * et remplacement par la méthode showAndWait()  
 **/  
primaryStage.showAndWait();  
/** Modification du texte avec la valeur retournée **/  
txfText.setText(controller.getTxfText());
```

Il nous faut également gérer le code de la méthode [evtOnMouseClickedBtnValider()] du contrôleur [SampleController2.java]. c'est le même que celle de la méthode [evtOnMouseClickedBtnAnnuler()]. Au lieu de dupliquer le code, il suffit d'appeler cette méthode.



```
/** Le traitement est le même que la méthode du  
 * bouton annuler, il suffit d'appeler cette méthode  
 **/  
@FXML private void evtOnMouseClickedBtnValider(){  
    evtOnMouseClickedBtnAnnuler();  
}
```

Testez votre programme :



3.3. Dernière amélioration, la gestion du bouton Annuler

Notre seconde fenêtre a deux boutons [**Annuler**] et [**Valider**].

- Le bouton [**Valider**] permet de renvoyer le texte modifié à la première fenêtre.
- Le bouton [**Annuler**] n devrait rien faire lui (*faites un test, ce n'est pas ce qui ce passe !!*)

En gardant notre logique Getters / Setters dans le second contrôleur, nous allons rajouter un nouvel attribut à notre contrôleur [`SampleController2.java`].

Nous allons le nommer [**isBtnValiderClicked**].



Le nom peut paraître long mais il est très explicite et nous n'avons pas à expliquer quelle est sa finalité.



```
/** Attributs de la classe */  
private boolean isBtnValiderClicked = false;
```

Cet attribut est par défaut initialisé à **false**. C'est un choix arbitraire, vous pouvez faire autrement.

Notre contrôleur appelant va analyser cette valeur pour savoir si le bouton [**Valider**] a été cliqué. Nous avons donc besoin d'un Getter.



```
public boolean getIsBtnValiderClicked() {  
    return isBtnValiderClicked;  
}
```

Il faut lui affecter la valeur **true** lorsque l'on clic sur le bouton [**Valider**]



```
@FXML private void evtOnMouseClickedBtnValider(){  
    isBtnValiderClicked = true;  
    evtOnMouseClickedBtnAnnuler();  
}
```

Il faut maintenant tester cette valeur dans notre premier constructeur.



```
/** Modification du texte avec la valeur retournée */  
if(controller.getIsBtnValiderClicked()) {  
    txfText.setText(controller.getTxfText());  
}
```

Vous pouvez tester entièrement votre application.