COURS DE JAVA - Interfaces Graphiques (Java Fx)

El Hadji Gaye - AlizNet

Avec Netbeans.

Auteur El Hadji Gaye

Pour Ecole

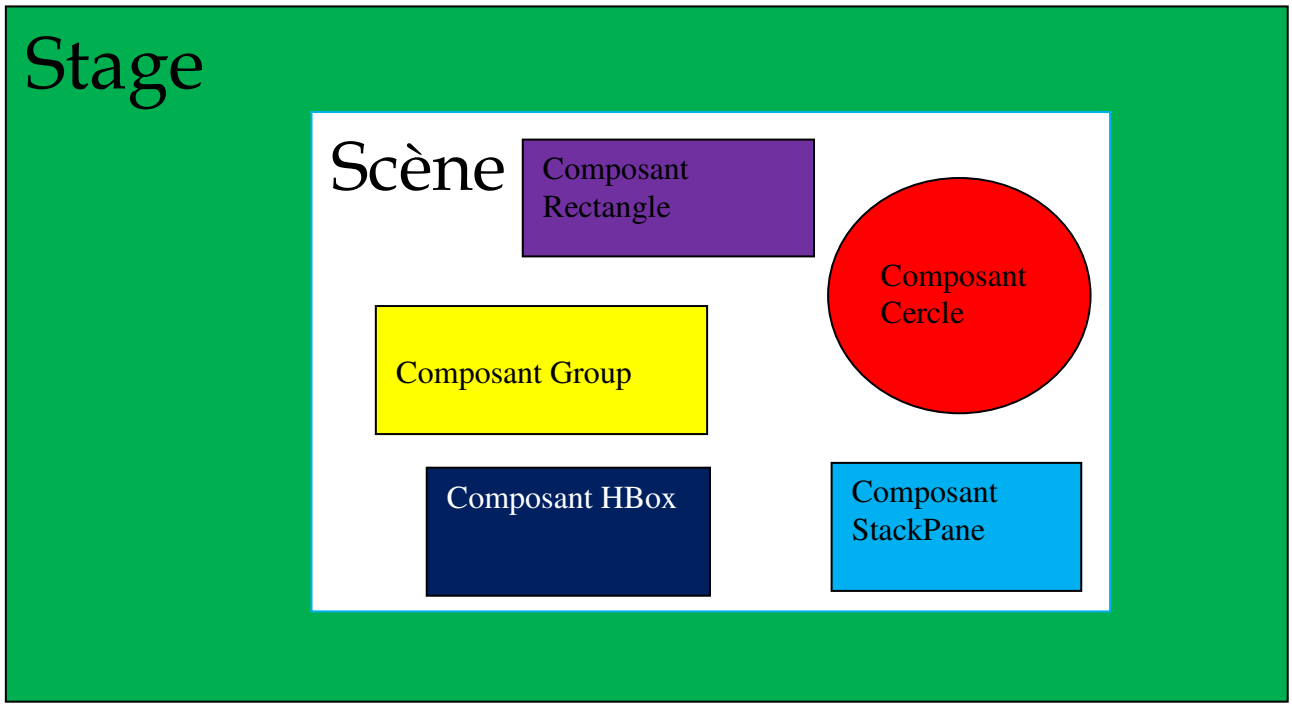
Date 11/02/19

Objet Cours de Java – Java Fx.

I) Généralités sur le Java Fx

Une application Java Fx est composée de trois choses essentielles :

- **Le Stage** : C'est un objet qui hérite de la classe « Window » qui est une fenêtre. Le stage est la fenêtre principale d'une application Java Fx c'est-à-dire l'espace dans lequel toutes les opérations applicatives s'effectueront.
- **La Scène** : Cet objet est contenu dans le Stage. Ce sera la scène dans laquelle notre application s'exécutera.
- **Les Nœuds** : Ce sont des éléments d'accroche de notre application c'est-à-dire les objets conteneurs dans lesquels nous allons accrocher tous les composants graphiques dans l'application Java Fx.
  Exemple de nœuds : StackPane, Group, VBox, HBox … etc.



Il sera possible de réaliser une application Java Fx de deux manières :

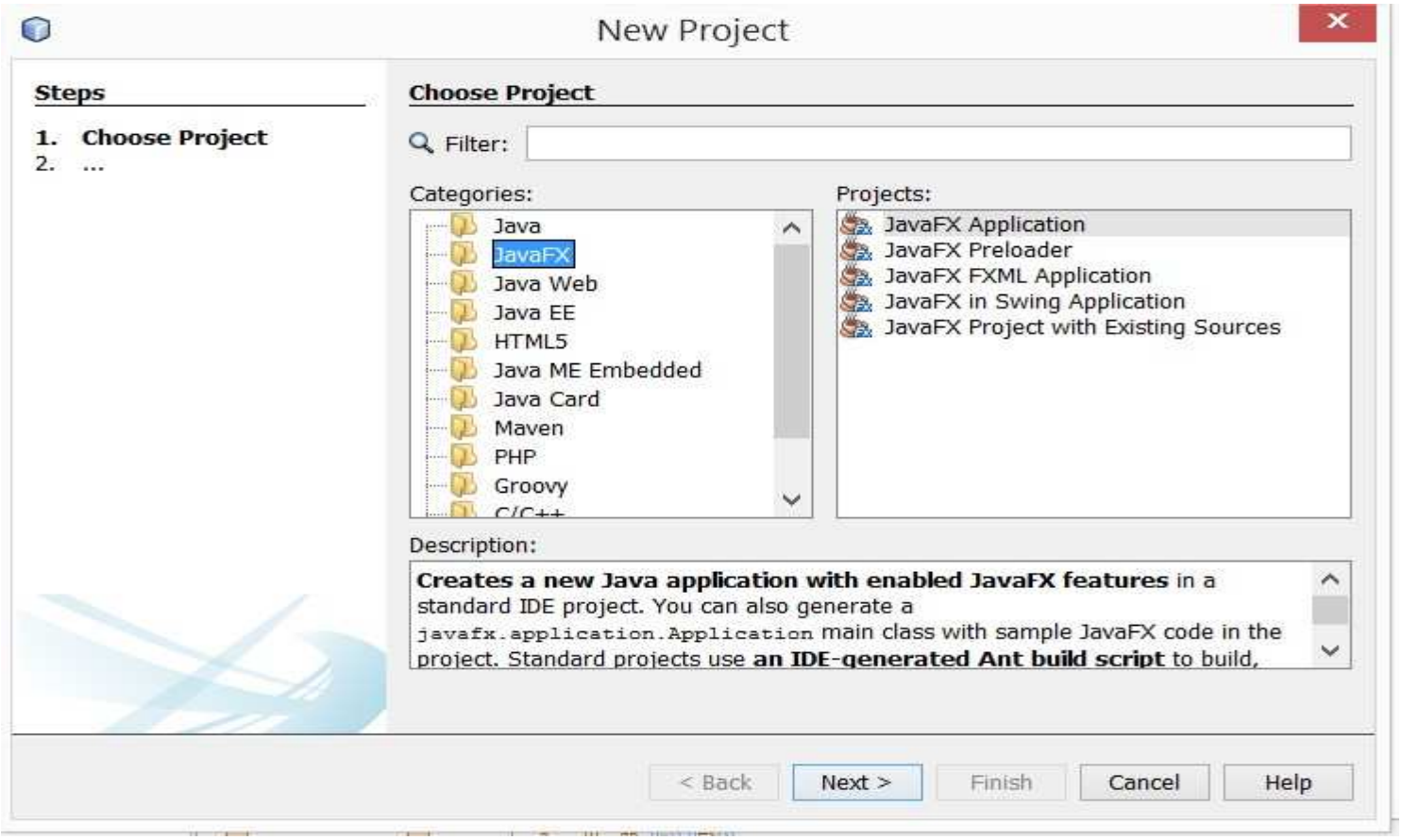
- Avec du code **100 % Java** c'est-à-dire les composants Java Fx seront instanciés avec du code Java.
- Avec un **éditeur Wysiwyg (Scène Builder)** qui permettra de créer les composants Java Fx sous forme de fichier **fxml** avec des balises Java Fx (ces balises ressemblent beaucoup à des balises HTML c'est-à-dire on aura des balises avec des attributs avec leurs valeurs). Les évènements liés aux composants seront par contre gérés en Java par des classes dites « contrôleurs ». Nous aurons aussi une liaison dites « binding » des composants **fxml** avec les classes contrôleurs.

Faire du Java Fx avec **Scène Builder** sera surement plus simple car on pourra positionner les éléments Java Fx plus facilement par Drag And Drop (Glisser et Déposer)

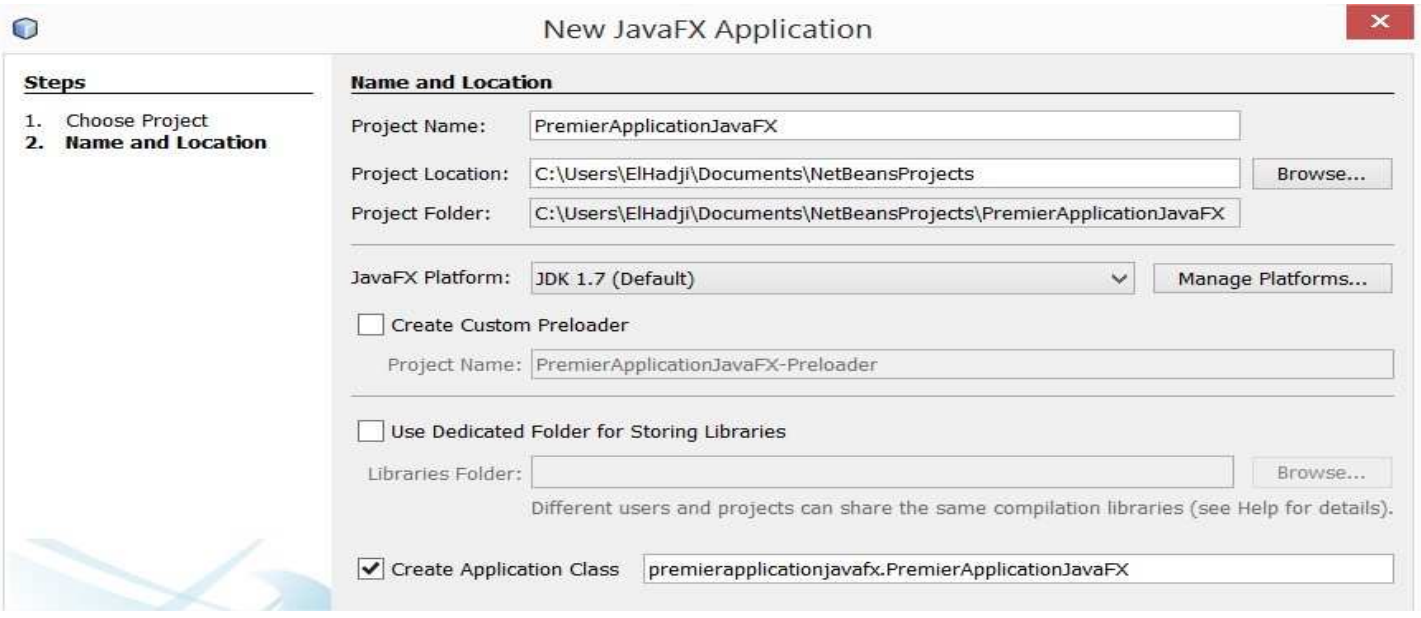
II) Mon premier Interface Java Fx

Créons notre première application Java Fx «PremierApplicationJavaFX».

Choisir « JavaFx » et « JavaFX Application ».



Mettre le nom de votre application Java Fx.



Ces différentes actions vont créer une classe de démarrage de l'application « PremierApplicationJavaFX.java ».

```
6   package premierapplicationjavafx;
7
8   import javafx.application.Application;
9   import javafx.event.ActionEvent;
10  import javafx.event.EventHandler;
11  import javafx.scene.Scene;
12  import javafx.scene.control.Button;
13  import javafx.scene.layout.StackPane;
14  import javafx.stage.Stage;
15
16  /**
17   *
18   * @author ElHadji
19   */
20  public class PremierApplicationJavaFX extends Application {
21
22      @Override
23      public void start(Stage primaryStage) {
24          Button btn = new Button();
25          btn.setText("Say 'Hello World'");
26          btn.setOnAction(new EventHandler<ActionEvent>() {
27
28              @Override
29              public void handle(ActionEvent event) {
30                  System.out.println("Hello World!");
31              }
32          });
33
34          StackPane root = new StackPane();
35          root.getChildren().add(btn);
36
37          Scene scene = new Scene(root, 300, 250);
38
39          primaryStage.setTitle("Hello World!");
40          primaryStage.setScene(scene);
41          primaryStage.show();
42      }
43
44      /**
45       * @param args the command line arguments
46       */
47      public static void main(String[] args) {
48          launch(args);
49      }
50
51  }
52
```

Ligne 24 : instanciation d'un objet de type « Button ».
Ligne 26 : création d'une action d'écoute sur le clic du composant « Button ».
Ligne 37 : instanciation de l'objet « Scene ».
Ligne 40 : affectation de la « Scene » au « Stage » **primaryStage**.
Ligne 41 : affichage de la fenêtre **primaryStage**.
A l'exécution de l'application on obtient :


Hello World!
Say 'Hello World'

III) Les évènements en Java Fx

Java Fx donnent la possibilité d'affecter à ces composants des événements.
L'ajout d'un évènement sur un composant se fait avec des objets de type « EventHandler ». Par exemple pour l'événement de clic sur un composant (Action) on aura :

```
monComposantFx.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("******* Evenement ActionEvent (Clic) ********");

        }
})
```

Pour l'évènement « **MousePressed**» qui correspond au clic de la souris on aura :

```
monComposantFx.setOnMousePressed(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
            System.out.println("******* Evenement MouseEvent ********");

        }
})
```

Pour l'évènement « **KeyPressed**» qui correspond au clic d'une touche on aura :

```
monComposantFx.setOnKeyPressed(new EventHandler< KeyEvent >() {
        @Override
        public void handle(KeyEvent event) {
            System.out.println("******* Evenement KeyEvent ********");

        }
})
```

Il existe encore plein d'autres évènements et leur utilisation est semblable au «**ActionEvent**», «**MousePressed** » et « **KeyPressed**». Par exemple : **OnMouseClicked** ,**OnDragDetected**, **OnDragDone**, **OnKeyReleased**, **OnRotate** etc. ……

On a aussi la possibilité de rajouter sur un composant un évènement d'écoute de changement d'état. L'ajout de cet écouteur spécial se fait avec la méthode « addListener ».

Pour le composant « TextField » on aura :

```
new TextField("Mon Texte ").textProperty().addListener(new ChangeListener<String>() {
        @Override
        public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
          System.out.println("oldValue: " + oldValue + " , newValue:" + newValue);
        }
    })
```

Pour le composant «ComboBox» on aura :

```
new ComboBox().getSelectionModel().selectedItemProperty().addListener(new ChangeListener<String>() {
        @Override
        public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
          System.out.println("Selected value: " + observableValue.getValue());
        }
    })
```

Pour le composant «TableView» on aura :

```
new TableView<Animal>().getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Animal>() {
        @Override
        public void changed(ObservableValue<? extends Animal> observable, Animal oldValue, Animal newValue) {
          System.out.println("Selected value: " + observable.getValue());
        }
    });
```

Pour le composant «RadioButton » on aura :

```
new ToggleGroup().selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
        @Override
        public void changed(ObservableValue<? extends Toggle> ov, Toggle oldToggle, Toggle newToggle) {
          RadioButton radioChecked = (RadioButton) oldToggle.getToggleGroup().getSelectedToggle();
          System.out.println("Selected Radio Button : " + radioChecked.getText());

        }
    })
```

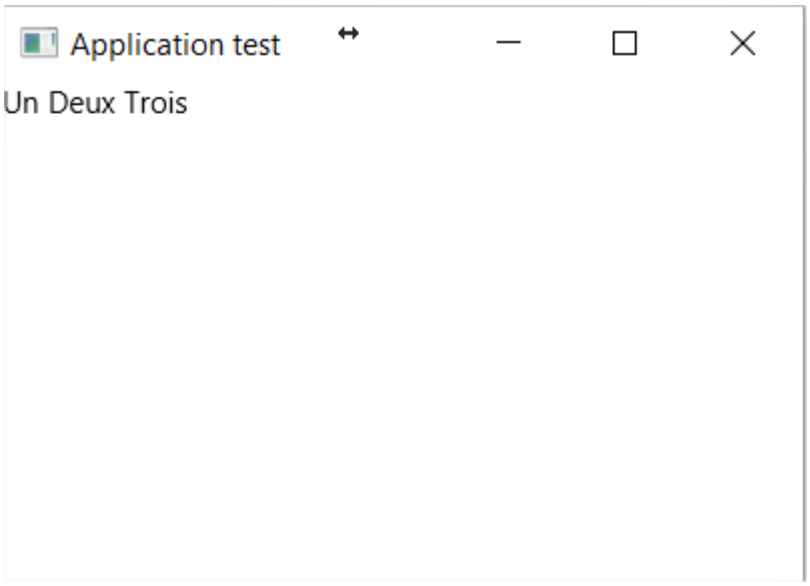
IV) Quelques composants du Java Fx

1) Composant « HBox»

Ce composant a pour chemin d'import « javafx.scene.layout.HBox ». Ce composant est un panel qui sert à mettre des éléments Java Fx de manière horizontale.

Exemple :

```
39
40          HBox hBox = new HBox();
41          hBox.getChildren().addAll(new Text("Un "), new Text("Deux "), new Text("Trois "));
42
43          StackPane root = new StackPane();
44          root.getChildren().add(hBox);
45
46          Scene scene = new Scene(root, 400, 250);
47          primaryStage.setTitle("Application test");
48          primaryStage.setScene(scene);
49          primaryStage.show();
```

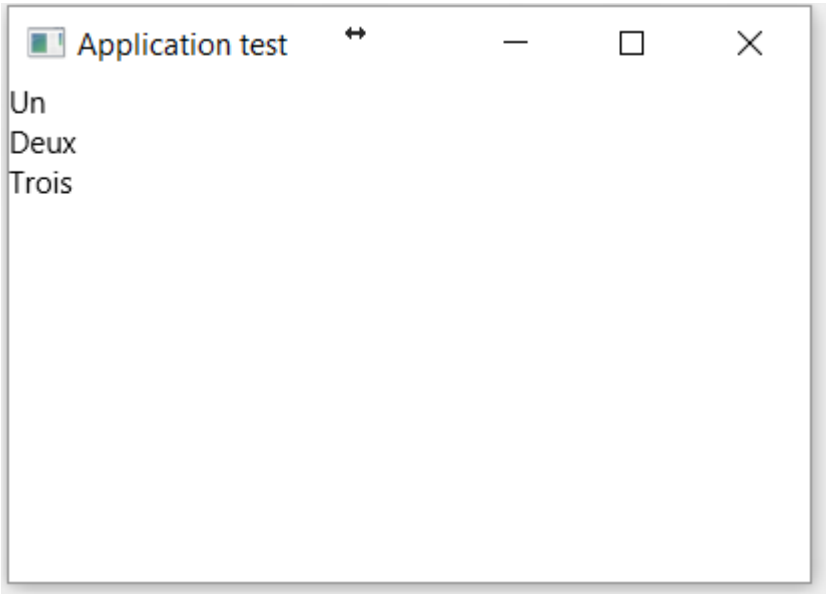


2) Composant « VBox»

Ce composant a pour chemin d'import « javafx.scene.layout.VBox ». Ce composant est un panel qui sert à mettre des éléments Java Fx de manière verticale.

Exemple :

```
40
41          VBox vBox = new VBox();
42          vBox.getChildren().addAll(new Text("Un "), new Text("Deux "), new Text("Trois "));
43
44          StackPane root = new StackPane();
45          root.getChildren().add(vBox);
46
47          Scene scene = new Scene(root, 400, 250);
48          primaryStage.setTitle("Application test");
49          primaryStage.setScene(scene);
50          primaryStage.show();
51
```



3) Composant « Button»

Ce composant a pour chemin d'import « javafx.scene.control.Button »

```
24          Button button = new Button();
25          button.setText("Mon Bouton");
26          button.setOnAction(new EventHandler<ActionEvent>() {
27              @Override
29              public void handle(ActionEvent event) {
30                  System.out.println("Clic sur Mon Bouton!");
31              }
            });
```

Ligne 24 : on instancie un objet de type « Button ».
Ligne 26 : on instancie un objet de type « ActionEvent » pour gérer l'événement du clic sur le bouton.

4) Composant « Text»

Ce composant a pour chemin d'import « javafx.scene.text.Text ». Il sert à afficher un label.

Exemple :

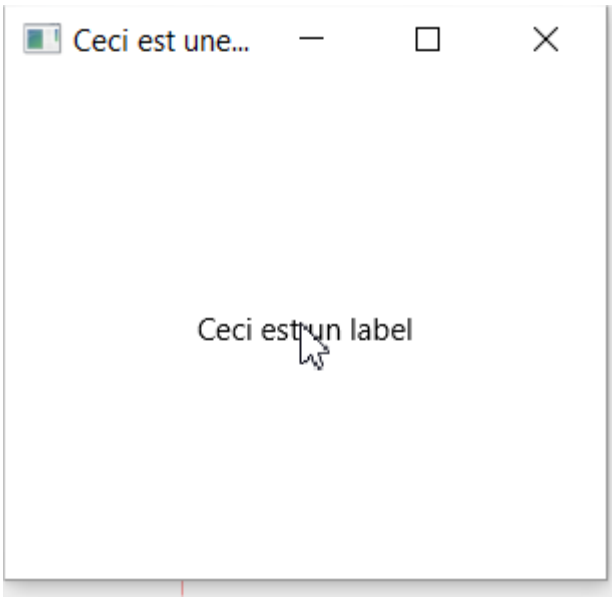
```
43
44          Text text = new Text("Ceci est un label");
45
46          StackPane root = new StackPane();
47          root.getChildren().add(text);
48
49          Scene scene = new Scene(root, 300, 250);
50          primaryStage.setTitle("Ceci est une application test");
51          primaryStage.setScene(scene);
52          primaryStage.show();
53
```

Ou

```
43
44          Text text = new Text();
45          text.setText("Ceci est un label");
46
47          StackPane root = new StackPane();
48          root.getChildren().add(text);
49
50          Scene scene = new Scene(root, 300, 250);
51          primaryStage.setTitle("Ceci est une application test");
52          primaryStage.setScene(scene);
53          primaryStage.show();
54
```

On peut aussi associer au composant un évènement par exemple l'évènement « MouseEvent » :

```
43
44      Text text = new Text("Ceci est un label");
45      text.setOnMousePressed( new EventHandler<MouseEvent>() {
46          @Override
47          public void handle(MouseEvent event) {
48              System.out.println("******* Evenement MouseEvent ******** EventType: "+event.getEventType());
49
50          }
51      });
```



Sur la console on a les logs ci-dessous lorsqu'on met le curseur de la souris sur le label que l'on vient de créer :

```
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
******* Evenement MouseEvent ******** EventType: MOUSE_PRESSED
```

5) Composant « TextField»

Ce composant a pour chemin d'import « javafx.scene.control.TextField ». Il sert à afficher un champ texte.

Exemple :

```
35
36          TextField textField = new TextField("valeur de mon texte");
37          StackPane root = new StackPane();
38          root.getChildren().add(textField);
39
40          Scene scene = new Scene(root, 400, 250);
41          primaryStage.setTitle("Application test");
42          primaryStage.setScene(scene);
43          primaryStage.show();
```

Ou

```
35
36          TextField textField = new TextField();
37          textField.setText("valeur de mon texte");
38
39          StackPane root = new StackPane();
40          root.getChildren().add(textField);
41
42          Scene scene = new Scene(root, 400, 250);
43          primaryStage.setTitle("Application test");
44          primaryStage.setScene(scene);
45          primaryStage.show();
```

On peut aussi associer au composant un évènement par exemple l'évènement « MouseEvent » :

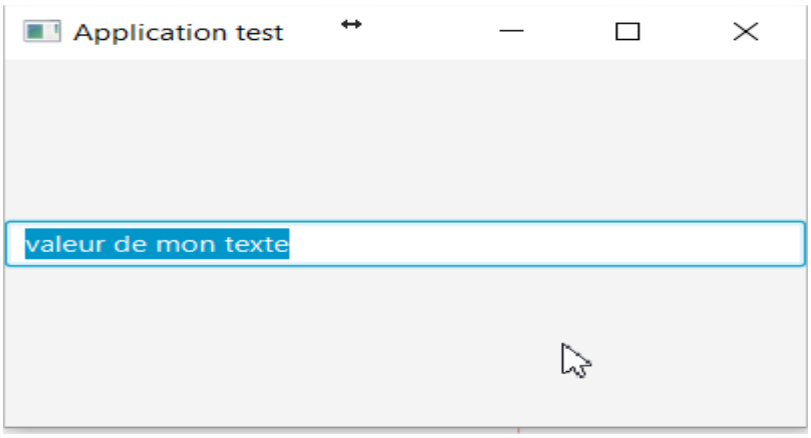
```
35
36      TextField textField = new TextField("valeur de mon texte");
37      textField.setOnMousePressed(new EventHandler<MouseEvent>() {
38          @Override
          public void handle(MouseEvent event) {
40              System.out.println("******* Evenement MouseEvent ******** EventType: " + event.getEventType());
41
42          }
43      });
```

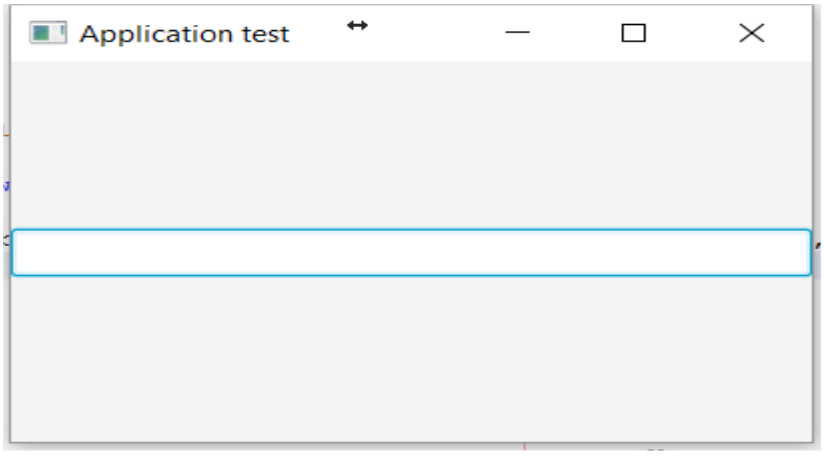
Ou l'évènement « MouseEvent » :

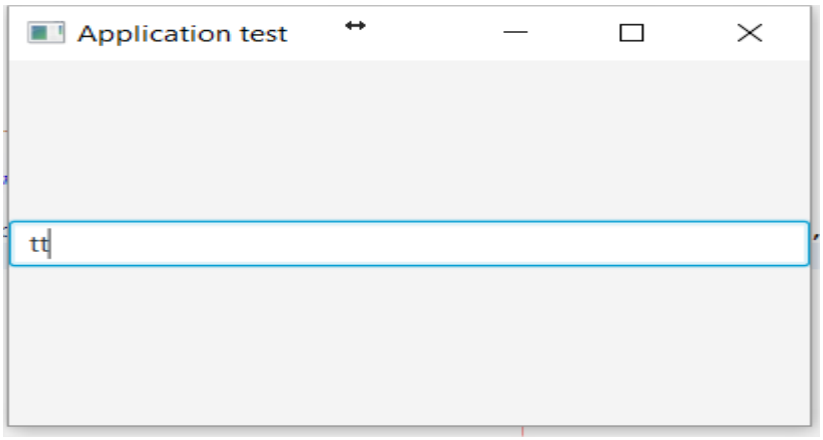
```
35
36          TextField textField = new TextField("valeur de mon texte");
37          textField.setOnKeyPressed(new EventHandler<KeyEvent>() {
38              @Override
39              public void handle(KeyEvent event) {
40                  System.out.println("******* Evenement KeyEvent ******** Text: " + event.getText());
41
42              }
43          });
44
```

On peut aussi ajouter un listener sur le changement d'état du composant :

```
41
42          textField.textProperty().addListener(new ChangeListener<String>() {
43              @Override
45              public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
46                  System.out.println("oldValue: " + oldValue + " , newValue:" + newValue);
47              }
48          });
```







La console affiche :

```
oldValue: valeur de mon texte , newValue:
oldValue:  , newValue:t
oldValue: t , newValue:tt
```

6) Composant « ComboBox »

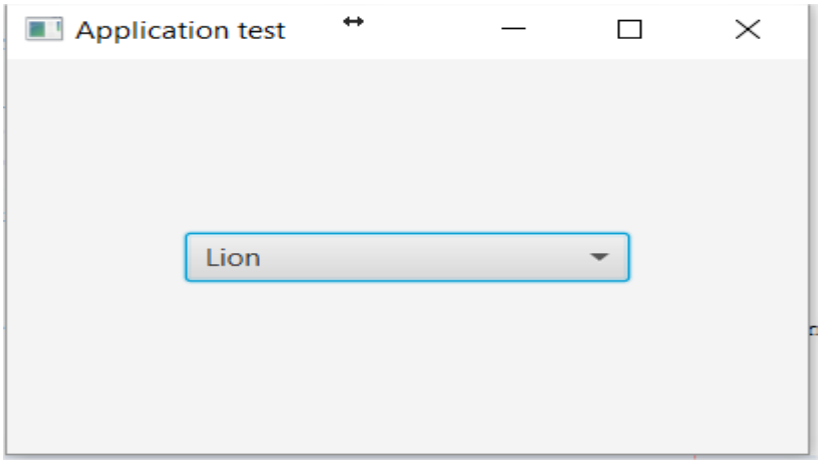
Ce composant a pour chemin d'import « javafx.scene.control.ComboBox ». Il sert à afficher des éléments avec plusieurs choix possibles.

Exemple :

```
39
40          ComboBox comboAnimaux = new ComboBox();
41
42          ObservableList<String> animaux = FXCollections.observableArrayList(
43                  "Chien", "Chat", "Lion", "Tigre", "Dauphin", "Requin", "Poule",
44                  "Mouton", "Lapin", "Loup", "Dinde", "Singe", "Elephant");
45          comboAnimaux.setPromptText("Selectionner votre animal");
46          comboAnimaux.setItems(animaux);
47
48          StackPane root = new StackPane();
49          root.getChildren().add(comboAnimaux);
50
51          Scene scene = new Scene(root, 400, 250);
52          primaryStage.setTitle("Application test");
53          primaryStage.setScene(scene);
54          primaryStage.show();
55
```

On peut utiliser le Listener « ChangeListener » pour détecter par exemple la sélection d'un élément dans le comboBox.

```
47
48          comboAnimaux.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<String>() {
49              @Override
50              public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
51                  System.out.println("selected value: " + observableValue.getValue());
52              }
53          });
```



On obtient alors sur la console :

```
selected value: Lion
```

Dans un évènement particulier on peut récuperer l'élèment qui a été sélectionné dans le ComboBox :

```
149
150         String selectedAnimal = (String) comboAnimaux.getSelectionModel().getSelectedItem();
151         System.out.println("selectedAnimal : " + selectedAnimal);
152
153
```

7) Composant « ListView »

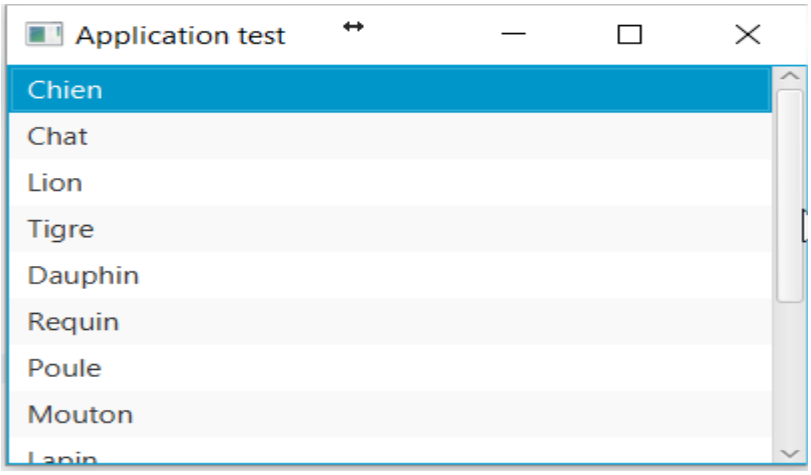
Ce composant a pour chemin d'import « javafx.scene.control.ListView ». Il sert à afficher des éléments avec plusieurs choix possibles, le choix multiple étant possible.

Exemple :

```
40
41          ListView listViewAnimaux = new ListView();
42
43          ObservableList<String> animaux = FXCollections.observableArrayList(
44                  "Chien", "Chat", "Lion", "Tigre", "Dauphin", "Requin", "Poule",
45                  "Mouton", "Lapin", "Loup", "Dinde", "Singe", "Elephant");
46          listViewAnimaux.setItems(animaux);
47
48
49          StackPane root = new StackPane();
50          root.getChildren().add(listViewAnimaux);
51
52          Scene scene = new Scene(root, 400, 250);
53          primaryStage.setTitle("Application test");
54          primaryStage.setScene(scene);
55          primaryStage.show();
```

On peut utiliser le Listener « ChangeListener » pour détecter par exemple la sélection d'un élèment dans le comboBox.

```
48
49          listViewAnimaux.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<String>() {
50              @Override
                public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
52                  System.out.println("Selected value: " + observableValue.getValue());
53              }
54          });
55
```



On obtient alors sur la console :

```
Selected value: Chien
```

Dans un évènement particulier on peut récupérer le ou les élèments qui ont été selectionnés dans le « ListView »:

```
169
170         String selectedAnimal = (String) listViewAnimaux.getSelectionModel().getSelectedItem();
171         ObservableList<String> selectedAnimals = listViewAnimaux.getSelectionModel().getSelectedItems();
172         System.out.println("selectedAnimal : " + selectedAnimal);
173         System.out.println("Les animaux Selectionnées : " + selectedAnimals);
174
```

8) Composant « TableView »

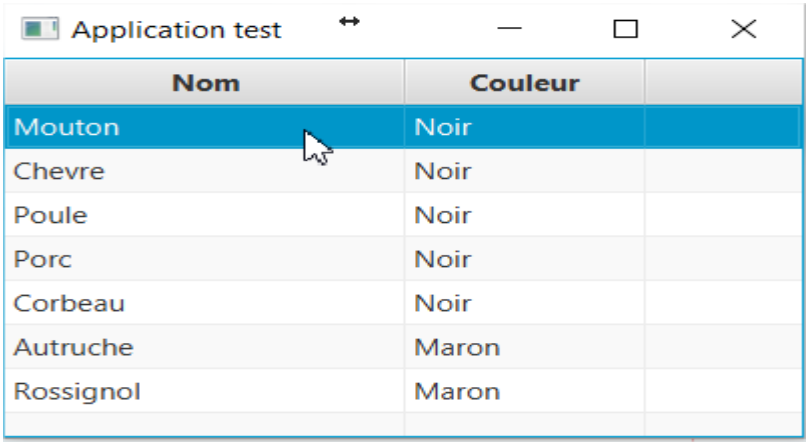
Ce composant a pour chemin d'import « javafx.scene.control.TableView ». Il sert à afficher un tableau.

Exemple :

```
46          TableView<Animal> tableView = new TableView<Animal>();
47          TableColumn nom = new TableColumn("Nom");
48          nom.setCellValueFactory(new PropertyValueFactory("nom"));
49          nom.setPrefWidth(200);
50
51          TableColumn couleur = new TableColumn("Couleur");
52          couleur.setCellValueFactory(new PropertyValueFactory("couleur"));
53          couleur.setPrefWidth(120);
54
55          ObservableList<Animal> animaux = FXCollections.observableArrayList(
56                  new Animal(2, "Mouton", 25, "23/12/2001", "Noir", 4, false),
57                  new Animal(3, "Chevre", 35, "21/12/2012", "Noir", 4, false),
58                  new Animal(4, "Poule", 1, "22/12/2009", "Noir", 2, false),
59                  new Animal(5, "Porc", 20, "23/12/2003", "Noir", 4, true),
60                  new Animal(6, "Corbeau", 5, "10/10/2013", "Noir", 2, true),
61                  new Animal(7, "Autruche", 5, "21/12/2014", "Maron", 2, false),
62                  new Animal(8, "Rossignol", 5, "23/12/2014", "Maron", 2, false));
63          tableView.getColumns().addAll(nom, couleur);
64          tableView.setItems(animaux);
65
66          StackPane root = new StackPane();
67          root.getChildren().add(tableView);
68
69          Scene scene = new Scene(root, 400, 250);
70          primaryStage.setTitle("Application test");
71          primaryStage.setScene(scene);
72          primaryStage.show();
73
```

On peut utiliser le Listener « ChangeListener » pour détecter par exemple la sélection d'un élèment dans le tableau.

```
65
66          tableView.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Animal>() {
67              @Override
69              public void changed(ObservableValue<? extends Animal> observable, Animal oldValue, Animal newValue) {
70                  System.out.println("Selected value: " + observable.getValue());
71              }
72          });
```



On obtient alors sur la console :

```
Selected value: [idAnimale=2, nom=Mouton, poids=25.0, dateNaissance=23/12/2001, couleur=Noir, nombrePattes=4, estCarnivore=false]
```

Dans un évènement particulier on peut recupérer le ou les élèments qui ont été sélectionnés dans le « TableView »:

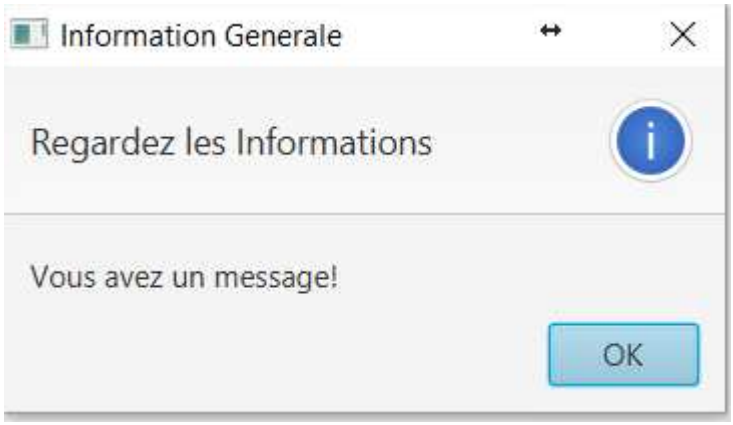
```
204
205         Animal selectedAnimal = (Animal) tableView.getSelectionModel().getSelectedItem();
206         ObservableList<Animal> selectedAnimals = tableView.getSelectionModel().getSelectedItems();
207         System.out.println("selectedAnimal : " + selectedAnimal);
208         System.out.println("Les animaux Selectionnées : " + selectedAnimals);
209
```

9) Composant « Alert »

Ce composant a pour chemin d'import « javafx.scene.control.Alert ». Il sert à afficher un Pop-Up.

Exemple :

```
46
47          Alert alert = new Alert(AlertType.INFORMATION);
48          alert.setTitle("Information Generale");
49          alert.setHeaderText("Regardez les Informations");
50          alert.setContentText("Vous avez un message!");
51          alert.showAndWait();
52
```



10) Composant « ToggleGroup »

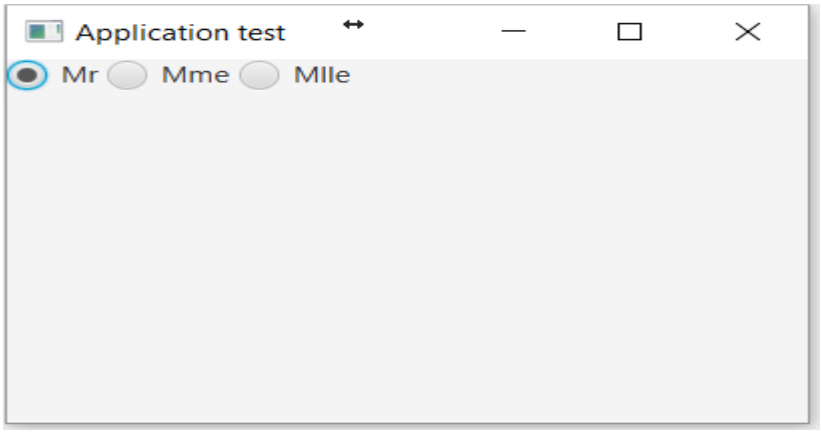
Ce composant a pour chemin d'import « javafx.scene.control.ToggleGroup ». Il sert à afficher un choix de boutons radio. Le composant bouton radio se nomme « RadioButton » et son chemin d'import est « javafx.scene.control.RadioButton ».

Exemple :

```
43
44          ToggleGroup toggleGroup = new ToggleGroup();
45
46          RadioButton radioMr = new RadioButton("Mr");
47          radioMr.setToggleGroup(toggleGroup);
48          radioMr.setSelected(true);
49
50          RadioButton radioMme = new RadioButton("Mme");
51          radioMme.setToggleGroup(toggleGroup);
52
53          RadioButton radioMlle = new RadioButton("Mlle");
54          radioMlle.setToggleGroup(toggleGroup);
55
56          HBox hbox = new HBox();
57          hbox.getChildren().addAll(radioMr, radioMme, radioMlle);
58
59          StackPane root = new StackPane();
60          root.getChildren().add(hbox);
61
62          Scene scene = new Scene(root, 400, 250);
63          primaryStage.setTitle("Application test");
64          primaryStage.setScene(scene);
65          primaryStage.show();
66
```

On peut utiliser le Listener « ChangeListener » pour detecter le changement de selection.

```
55
56          toggleGroup.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
57              @Override
              public void changed(ObservableValue<? extends Toggle> ov, Toggle oldToggle, Toggle newToggle) {
59                  RadioButton radioChecked = (RadioButton) oldToggle.getToggleGroup().getSelectedToggle();
60                  System.out.println("Selected Radio Button : " + radioChecked.getText());
61
62              }
63          });
```



On obtient alors sur la console :

```
Selected Radio Button : Mme
Selected Radio Button : Mlle
Selected Radio Button : Mme
```

11) Composant « CheckBox »

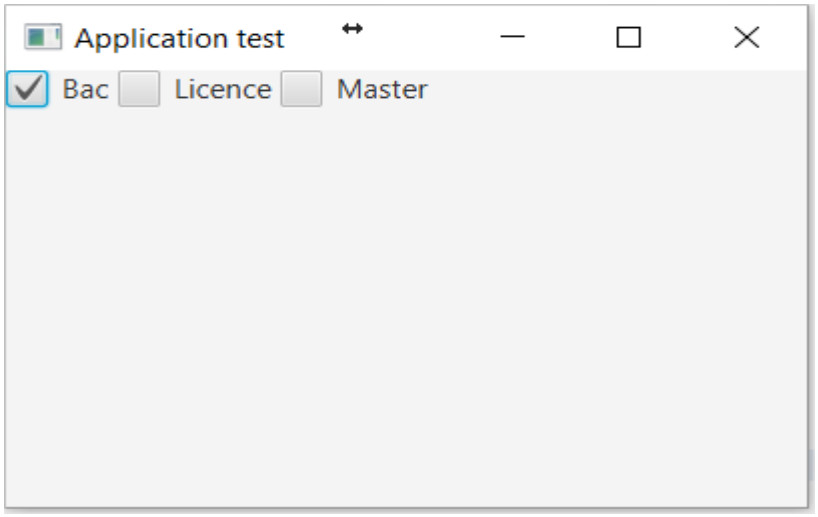
Ce composant a pour chemin d'import « javafx.scene.control.CheckBox ».

Exemple :

```
        HBox hBox = new HBox();
        CheckBox checkBoxBac = new CheckBox("Bac ");
        CheckBox checkBoxLicence = new CheckBox("Licence ");
        CheckBox checkBoxMaster = new CheckBox("Master ");
        checkBoxBac.setSelected(true);

        hBox.getChildren().addAll(checkBoxBac, checkBoxLicence, checkBoxMaster);

        StackPane root = new StackPane();
        root.getChildren().add(createCheckBoxElement());

        Scene scene = new Scene(root, 400, 250);
        primaryStage.setTitle("Application test");
        primaryStage.setScene(scene);
        primaryStage.show();
```

On peut utiliser le Listener « ChangeListener » pour détecter par exemple la sélection d'un élèment dans le comboBox.

```
        checkBoxBac.selectedProperty().addListener(new ChangeListener<Boolean>() {
            @Override
            public void changed(ObservableValue<? extends Boolean> observable, Boolean oldValue, Boolean newValue) {
                System.out.println("newValue : " + newValue);
            }
        });
```

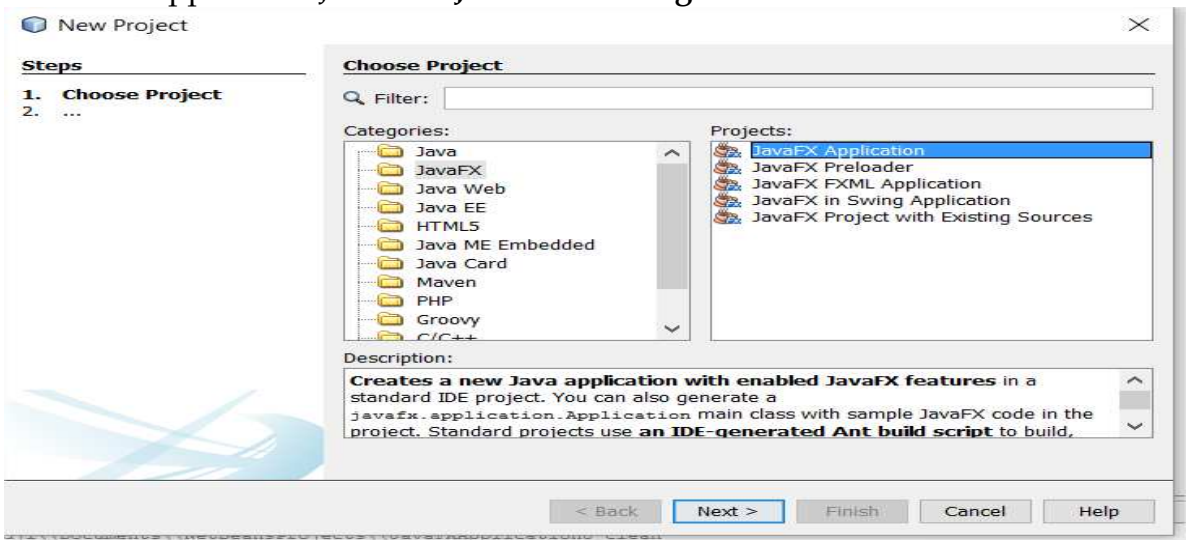


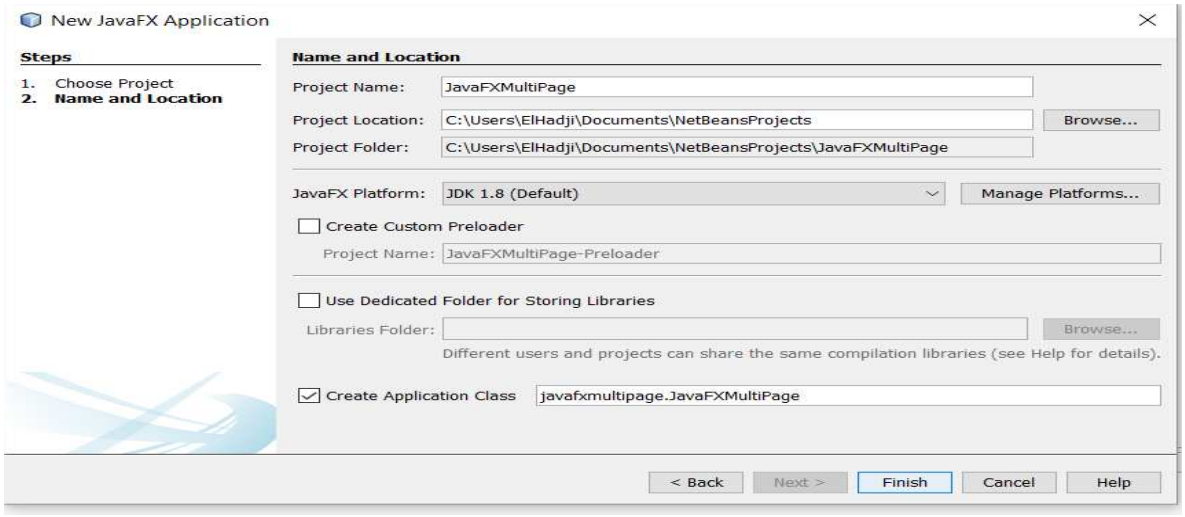
On obtient alors sur la console :

```
newValue : false
newValue : true
```

V) Concevoir une application Multipages avec Java Fx

Le but de cette partie du cours est de voir comment construire une application avec plusieurs contenus ou page.
Créer une application Java Fx « **JavaFXMultiPage** ».






```
20  public class JavaFXMultiPage extends Application {
21
22      @Override
23      public void start(Stage primaryStage) {
24          Button btn = new Button();
25          btn.setText("Say 'Hello World'");
26          btn.setOnAction(new EventHandler<ActionEvent>() {
27
28              @Override
29              public void handle(ActionEvent event) {
30                  System.out.println("Hello World!");
31              }
32          });
33
34          StackPane root = new StackPane();
35          root.getChildren().add(btn);
36
37          Scene scene = new Scene(root, 300, 250);
38
39          primaryStage.setTitle("Hello World!");
40          primaryStage.setScene(scene);
41          primaryStage.show();
42      }
43
44      /**
45       * @param args the command line arguments
46       */
47      public static void main(String[] args) {
48          launch(args);
49      }
50
51  }
```

Nous allons concevoir une application multipages avec les pages suivantes :

- **Page principale** : MainPage.java
- **Page rectangle** : RectanglePage.java
- **Page carré** : SquarePage.java.
- **Page cercle** : CirclePage.java.

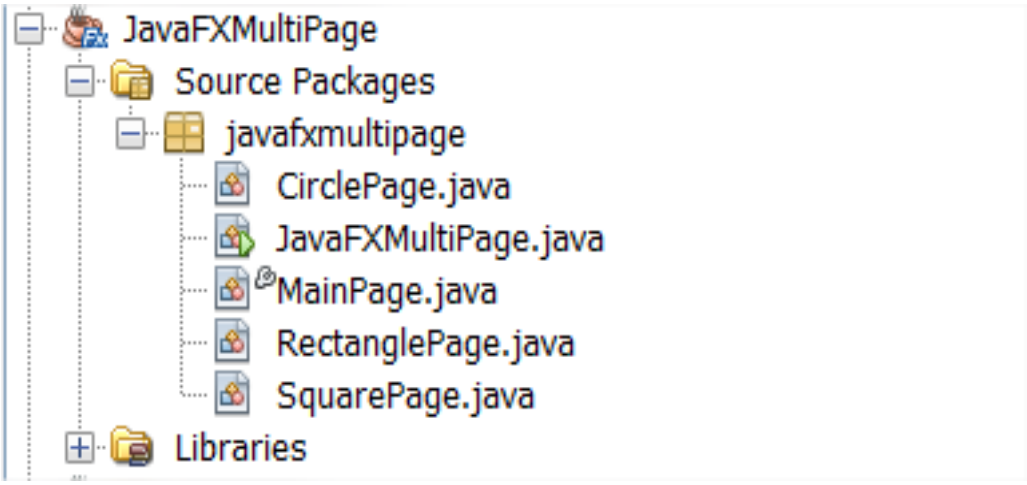
Les classes « **MainPage** », « **RectanglePage** », « **SquarePage** » et « **CirclePage** » héritent toutes de « **javafx.scene.layout.VBox** » elles sont donc par conséquent toutes des « **VBox** ».

Dans la fenêtre principale (**MainPage**) nous avons :

- **Un bloc Header :** avec les boutons « Quitter l'application », « Header Content 1 » et « Header Content 2 ».
- **Un bloc Body :** avec initialement un bouton « **Afficher RectanglePage** » et d'autres contenus par la suite.
- **Un bloc Footer :** avec les boutons « Quitter l'application », « Footer Content 1 » et « Footer Content 2 ».

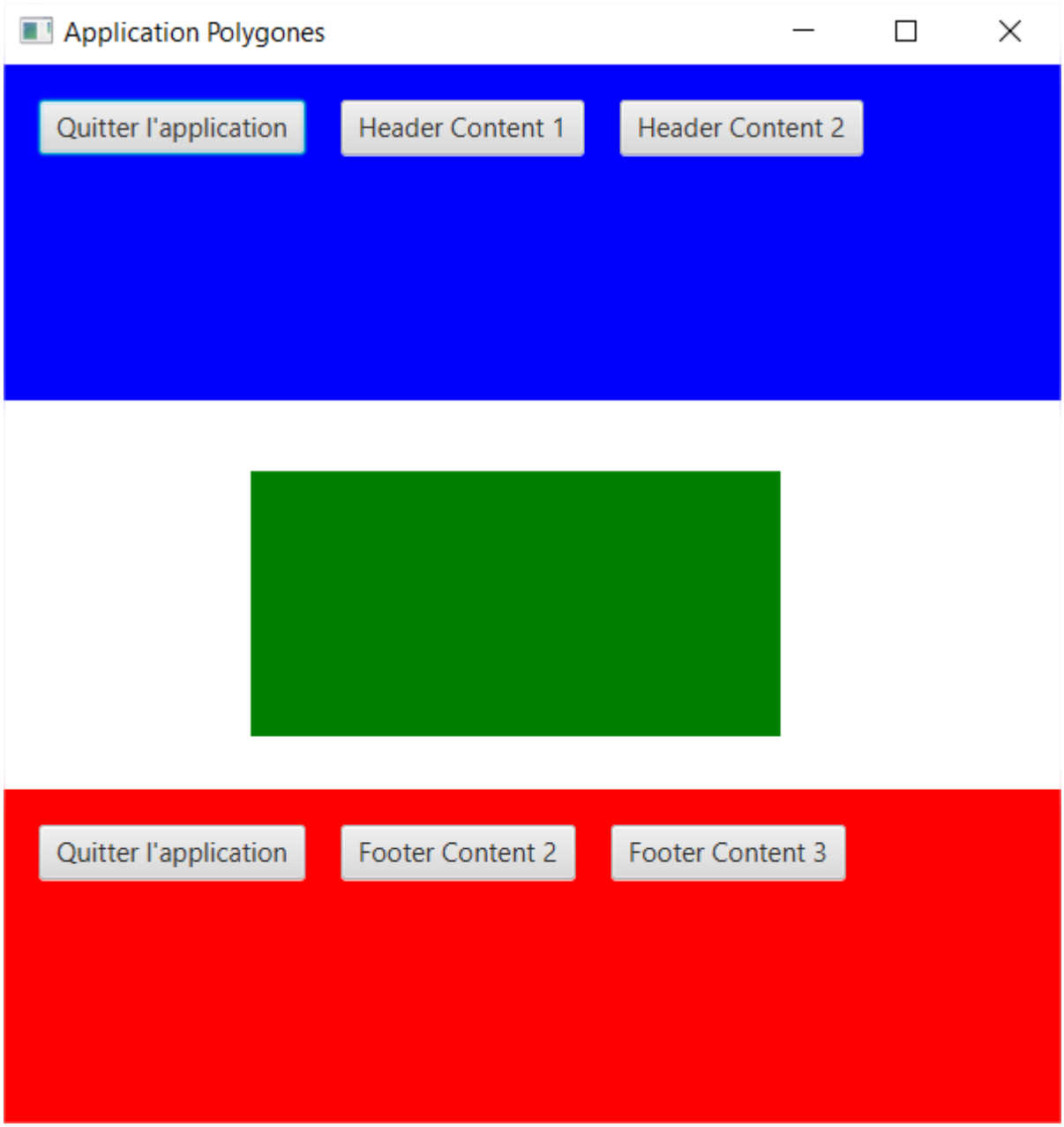
Au démarrage de l'application nous allons afficher « **MainPage** » dans lequel nous avons le bouton « **Afficher RectanglePage** » pour afficher la page « **RectanglePage** », puis dans « **RectanglePage** » au clic sur le rectangle nous afficherons « **SquarePage** » et enfin dans « **SquarePage** » au clic sur le carré nous afficherons « **CirclePage** » qui affichera un cercle.

Voici l'arborescence du projet :

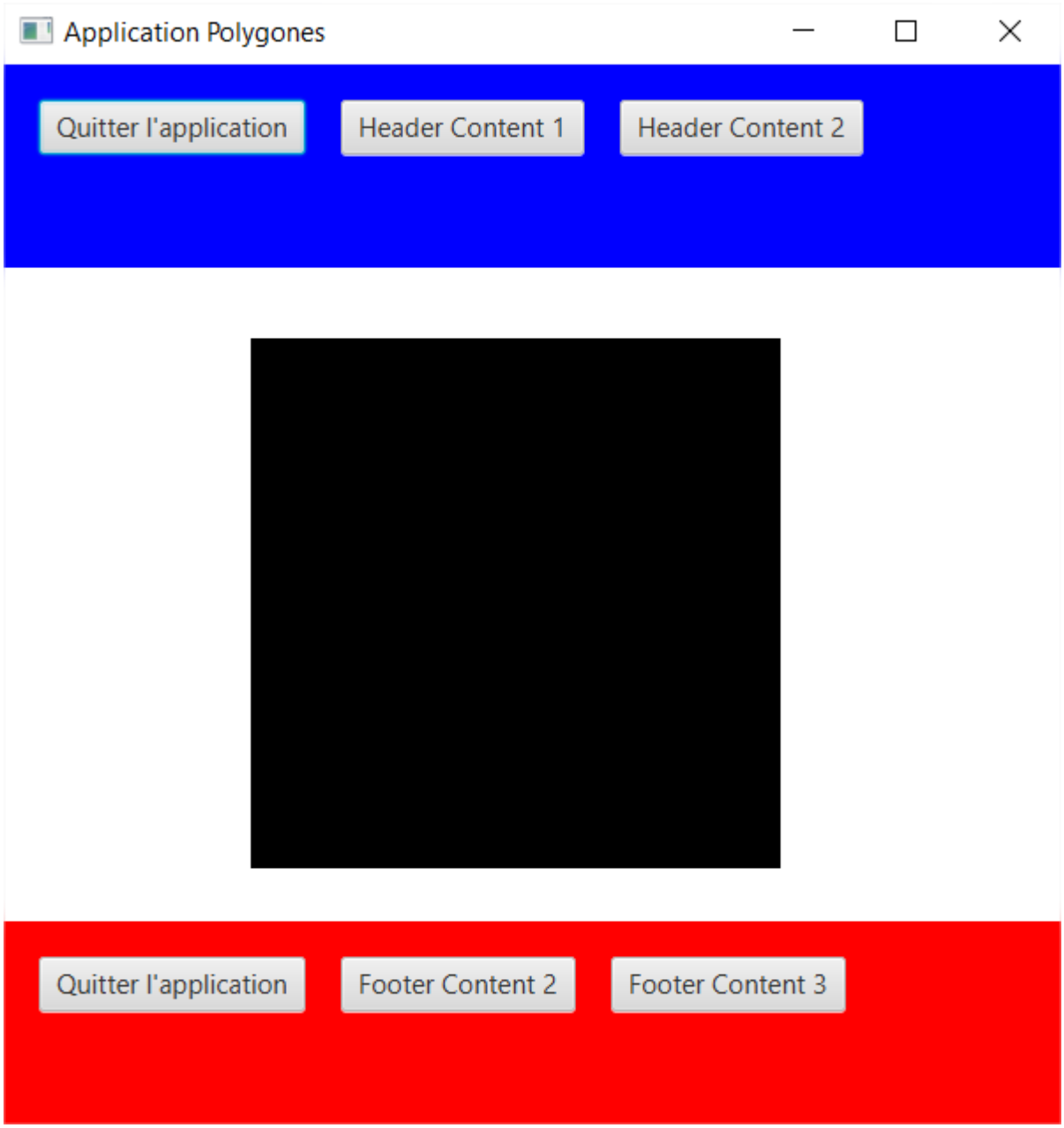


Au démarrage de l'application nous aurons :

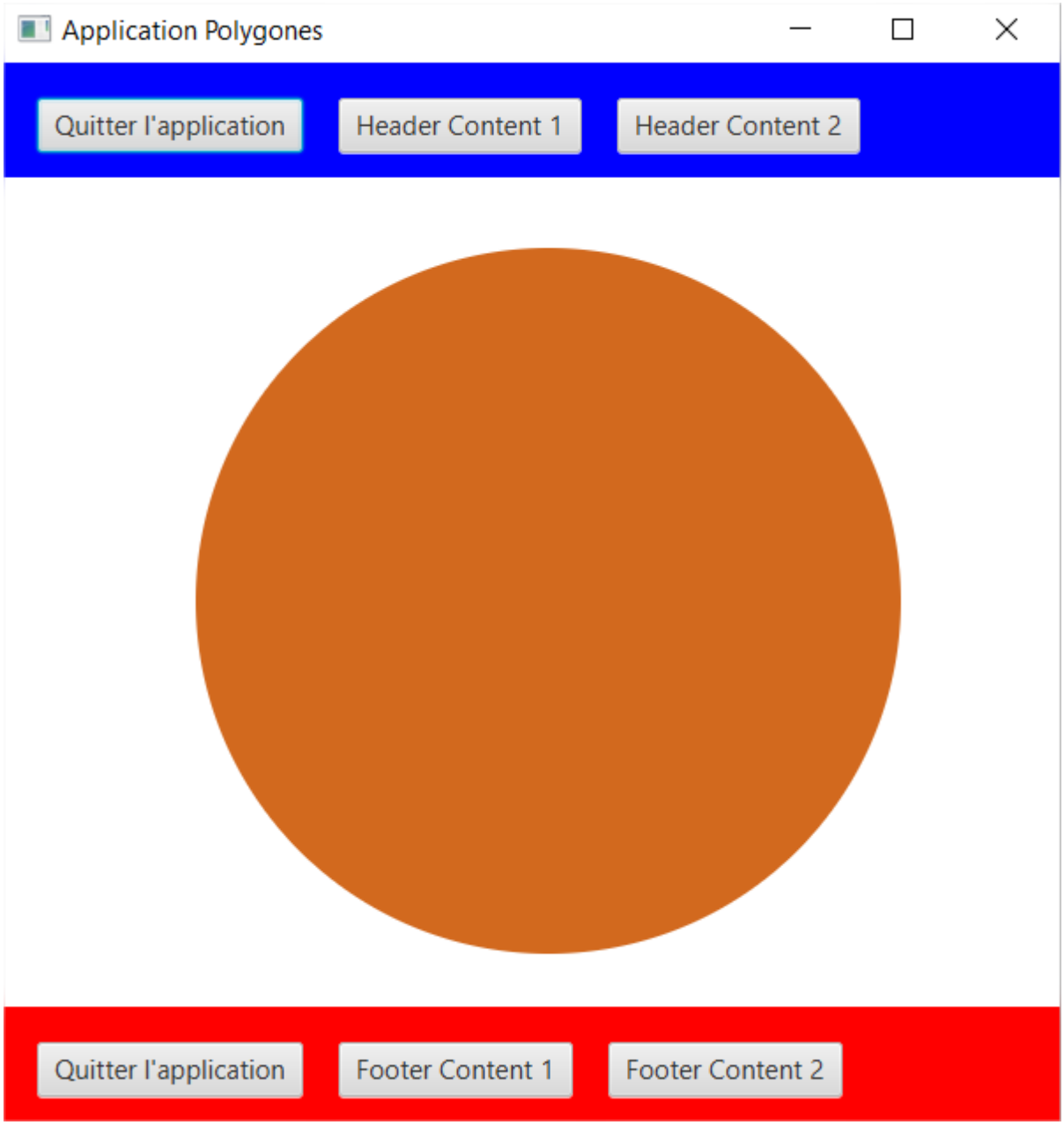
Au clic sur le bouton « **Afficher Rectangle** » :



Au clic sur le rectangle de « **RectanglePage** » :



Enfin au clic sur le Carré de « **SquarePage** » :



La classe « **JavaFXMultiPage** » devient donc :

```
6   package javafxmultipage;
7
8   import javafx.application.Application;
9   import javafx.scene.Scene;
10  import javafx.stage.Stage;
11
12  /**
13   *
14   * @author ElHadji
15   */
16  public class JavaFXMultiPage extends Application {
17
18      @Override
19      public void start(Stage primaryStage) {
20          MainPage mainPage = new MainPage();
21          Scene scene = new Scene(mainPage, 500, 500);
22          primaryStage.setTitle("Application Polygones");
23          primaryStage.setScene(scene);
24          primaryStage.show();
25      }
26
27      /**
28       * @param args the command line arguments
29       */
30      public static void main(String[] args) {
31          launch(args);
32      }
33  }
34
```

Le prototype de « **MainPage** » peut ressembler à :

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javafxmultipage;

import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

/**
 *
 * @author ElHadji
 */
public class MainPage extends VBox {

    private HBox headerContent = new HBox();
    private HBox bodyContent = new HBox();
    private HBox footerContent = new HBox();

    public MainPage() {
        super();
        buildHeader();
        buildBody();
        buildFooter();
    }

    private void buildHeader() {
    }

    private void buildBody() {
    }

    private void buildFooter() {
    }

    public HBox getHeaderContent() {
        return headerContent;
    }

    public void setHeaderContent(HBox headerContent) {
        this.headerContent = headerContent;
    }

    public HBox getBodyContent() {
        return bodyContent;
    }

    public void setBodyContent(HBox bodyContent) {
        this.bodyContent = bodyContent;
    }

    public HBox getFooterContent() {
        return footerContent;
    }

    public void setFooterContent(HBox footerContent) {
        this.footerContent = footerContent;
    }
}
```

Nous aurons besoin depuis « **RectanglePage** », « **SquarePage** » et « **CirclePage** » de modifier l'attribut « **bodyContent** » de « **MainPage** », ce qui implique que « **RectanglePage** », « **SquarePage** » et « **CirclePage** » doivent tous avoir comme attribut « **MainPage** ».

Le prototype de « **RectanglePage** » peut être donc :

```
6   package javafxmultipage;
7
8   import javafx.scene.layout.VBox;
9
10  /**
11   *
12   * @author ElHadji
13   */
14  public class RectanglePage extends VBox {
15
16      private MainPage mainPage = null;
17
18      public RectanglePage() {
19          super();
20          buildRectangle();
21      }
22
23      public RectanglePage(MainPage main) {
24          super();
25          this.mainPage = main;
26          buildRectangle();
27      }
28
29      private void buildRectangle() {
30      }
31
32      public MainPage getMainPage() {
33          return mainPage;
34      }
35  }
36
```

Le prototype de « **SquarePage** » :

```
6   package javafxmultipage;
7
8   import javafx.scene.layout.VBox;
9
10  /**
11   *
12   * @author ElHadji
13   */
14  public class SquarePage extends VBox {
15
16      private MainPage mainPage = null;
17
18      public SquarePage() {
19          super();
20          buildSquarePage();
21      }
22
23      public SquarePage(MainPage main) {
24          super();
25          this.mainPage = main;
26          buildSquarePage();
27      }
28
29      private void buildSquarePage() {
30      }
31
32      public MainPage getMainPage() {
33          return mainPage;
34      }
35  }
36
```

Le prototype de « **CirclePage** » :

```
6  package javafxmultipage;
7
8  import javafx.scene.layout.VBox;
9
10 /**
11  *
12  * @author ElHadji
13  */
14 public class CirclePage extends VBox {
15
16     private MainPage mainPage = null;
17
18     public CirclePage() {
19         super();
20         buildCirclePage();
21     }
22
23     public CirclePage(MainPage main) {
24         super();
25         this.mainPage = main;
26         buildCirclePage();
27     }
28
29     private void buildCirclePage() {
30     }
31
32     public MainPage getMainPage() {
33         return mainPage;
34     }
35 }
36
```

En définitive les méthodes « **buildHeader** », « **buildBody** » et « **buildFooter**» de « **MainPage** » peuvent avoir comme signature :

buildHeader ( ) :

```
32
33      private void buildHeader() {
34          Button exit = new Button("Quitter l'application");
35          Button content1 = new Button("Header Content 1");
36          Button content2 = new Button("Header Content 2");
37
38          Text separator1 = new Text("");
39          separator1.setWrappingWidth(20);
40          Text separator2 = new Text("");
41          separator2.setWrappingWidth(20);
42          Text separator3 = new Text("");
43          separator3.setWrappingWidth(20);
44
45          exit.setOnAction(new EventHandler<ActionEvent>() {
46              @Override
47              public void handle(ActionEvent event) {
48                  Platform.exit();
49              }
50          });
51          headerContent.getChildren().addAll(exit, separator1, content1, separator2, content2, separator3);
52          headerContent.setStyle("-fx-padding: 20 0 0 20;-fx-background-color: blue;");
53          headerContent.setPrefWidth(200);
54          headerContent.setPrefHeight(200);
55          this.getChildren().addAll(headerContent);
56      }
57
```

Ligne 48 : Exit de l'application.

buildBody ( ) :

```
57
58      private void buildBody() {
59          Button displayRectangle = new Button("Afficher RectanglePage");
60          displayRectangle.setPrefWidth(200);
61          displayRectangle.setPrefHeight(20);
62          bodyContent.getChildren().addAll(displayRectangle);
63          bodyContent.setStyle("-fx-padding: 20 0 0 20; -fx-background-color: white;");
64          bodyContent.setPrefWidth(200);
65          bodyContent.setPrefHeight(200);
66
67          displayRectangle.setOnAction(new EventHandler<ActionEvent>() {
68              @Override
69              public void handle(ActionEvent event) {
70                  // MainPage.this car this est une instance de EventHandler dans la methode handle
71                  RectanglePage rectanglePage = new RectanglePage(MainPage.this);
72                  // On vide le contenu body de MainPage et on y met RectanglePage
73                  getBodyContent().getChildren().clear();
74                  getBodyContent().getChildren().add(rectanglePage);
75              }
76          });
77          this.getChildren().addAll(bodyContent);
78      }
```

Ligne 71 : instanciation du **RectanglePage**.

buildFooter ( ) :

```
79
80      private void buildFooter() {
81          Button exit = new Button("Quitter l'application");
82          Button content1 = new Button("Footer Content 1");
83          Button content2 = new Button("Footer Content 2");
84
85          Text separator1 = new Text("");
86          separator1.setWrappingWidth(20);
87          Text separator2 = new Text("");
88          separator2.setWrappingWidth(20);
89          Text separator3 = new Text("");
90          separator3.setWrappingWidth(20);
91
92          exit.setOnAction(new EventHandler<ActionEvent>() {
93              @Override
95              public void handle(ActionEvent event) {
96                  Platform.exit();
97              }
98          });
99
100         footerContent.getChildren().addAll(exit, separator1, content1, separator2, content2, separator3);
101         footerContent.setStyle("-fx-padding: 20 0 0 20;-fx-background-color: red;");
102         footerContent.setPrefWidth(200);
103         footerContent.setPrefHeight(200);
104         this.getChildren().addAll(footerContent);
105     }
```

Ligne 95 : Exit de l'application.

La méthode « **buildRectangle** » de « **RectanglePage** » peut avoir comme signature :

```
32
33      private void buildRectangle() {
34          Rectangle rectangle = new Rectangle(100, 100, 300, 150);
35          rectangle.setFill(Color.GREEN);
36          rectangle.setOnMouseClicked(new EventHandler<MouseEvent>() {
37              @Override
39              public void handle(MouseEvent event) {
40                  SquarePage squarePage = new SquarePage(mainPage);
41                  // On vide le contenu body de MainPage et on y met SquarePage
42                  mainPage.getBodyContent().getChildren().clear();
43                  mainPage.getBodyContent().getChildren().add(squarePage);
44
45              }
46          });
47          this.setStyle("-fx-padding: 20 20 30 120;");
48          this.getChildren().add(rectangle);
49      }
```

Ligne 36 : Création d'un évènement sur le clic du rectangle.

La méthode « **buildSquarePage** » de « **RectanglePage** » peut avoir comme signature :

```
32
33      private void buildSquarePage() {
34          Rectangle square = new Rectangle(100, 100, 300, 300);
35          square.setFill(Color.BLACK);
36          square.setOnMouseClicked(new EventHandler<MouseEvent>() {
37              @Override
38              public void handle(MouseEvent event) {
39                  CirclePage circlePage = new CirclePage(mainPage);
40                  // On vide le contenu body de MainPage et on y met SquarePage
41                  mainPage.getBodyContent().getChildren().clear();
42                  mainPage.getBodyContent().getChildren().add(circlePage);
43
44              }
45          });
46          this.setStyle("-fx-padding: 20 20 30 120;");
47          this.getChildren().add(square);
48      }
49
```

Ligne 36 : Création d'un événement sur le clic du carée.

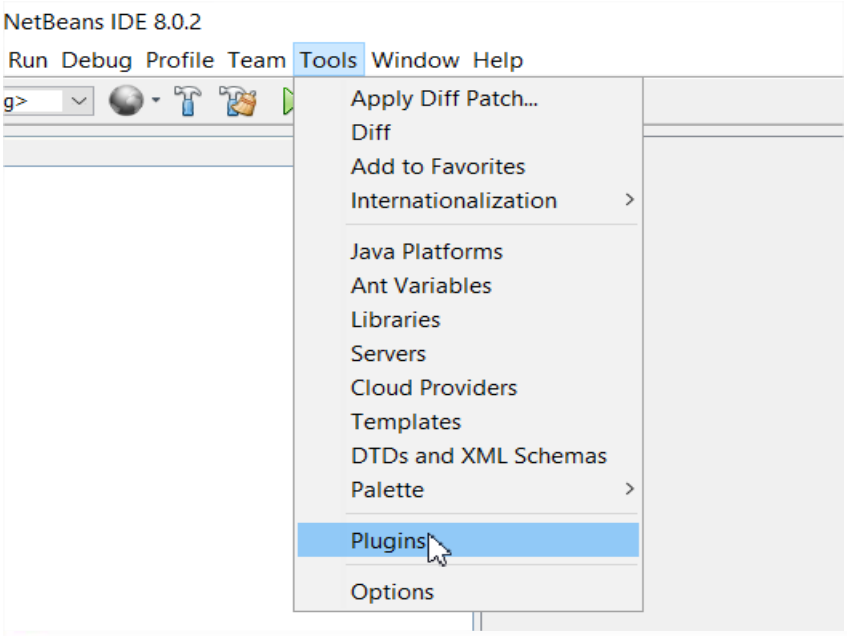
La méthode « **buildCirclePage** » de « **CirclePage** » peut avoir comme signature :

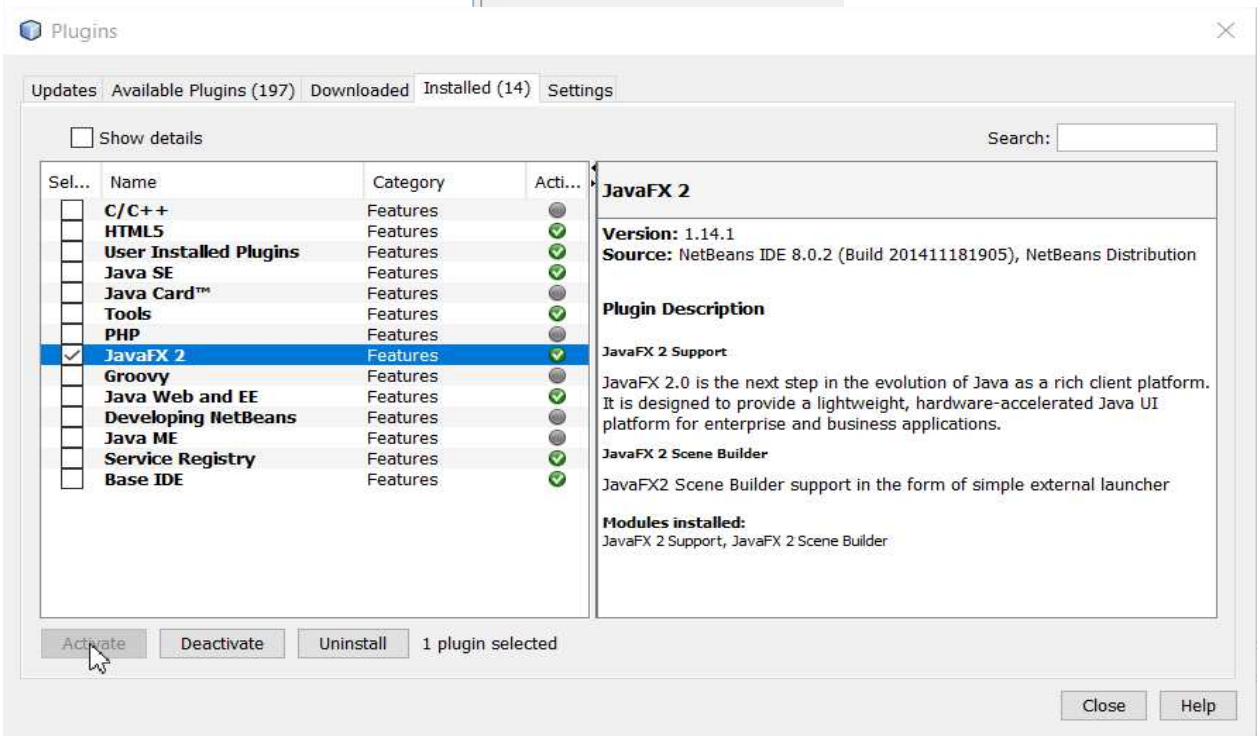
```
30
31      private void buildCirclePage() {
32          Circle circle = new Circle(100, 100, 200);
33          circle.setFill(Color.CHOCOLATE);
34          this.setStyle("-fx-padding: 20 20 30 90;");
35          this.getChildren().add(circle);
36      }
```

VI) Installation de Scène Builder

Afin de simplifier le développement Java Fx il est plus facile d'utiliser un outil wiziwig qui permet de glisser, déposer et positionner les éléments de notre page.

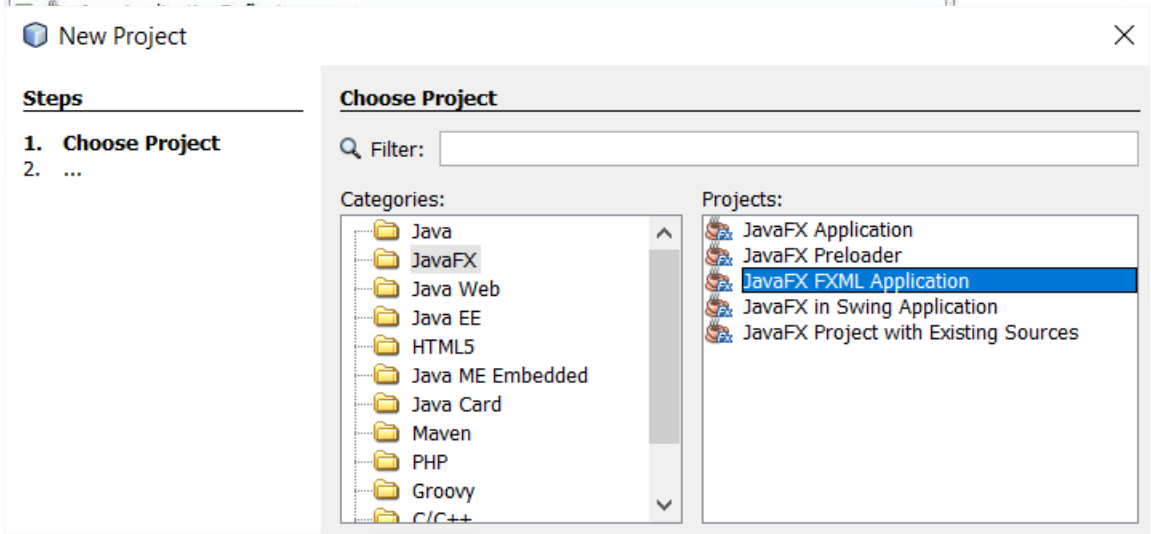
Installer **Java Fx** sur http://www.oracle.com/technetwork/java/javafx2-archive-download-1939373.html.
Installer le **Scène Builder 2** sur http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html.
Une fois que vous avez fini d'installer Scène Builder, activer Java Fx sur NetBeans.

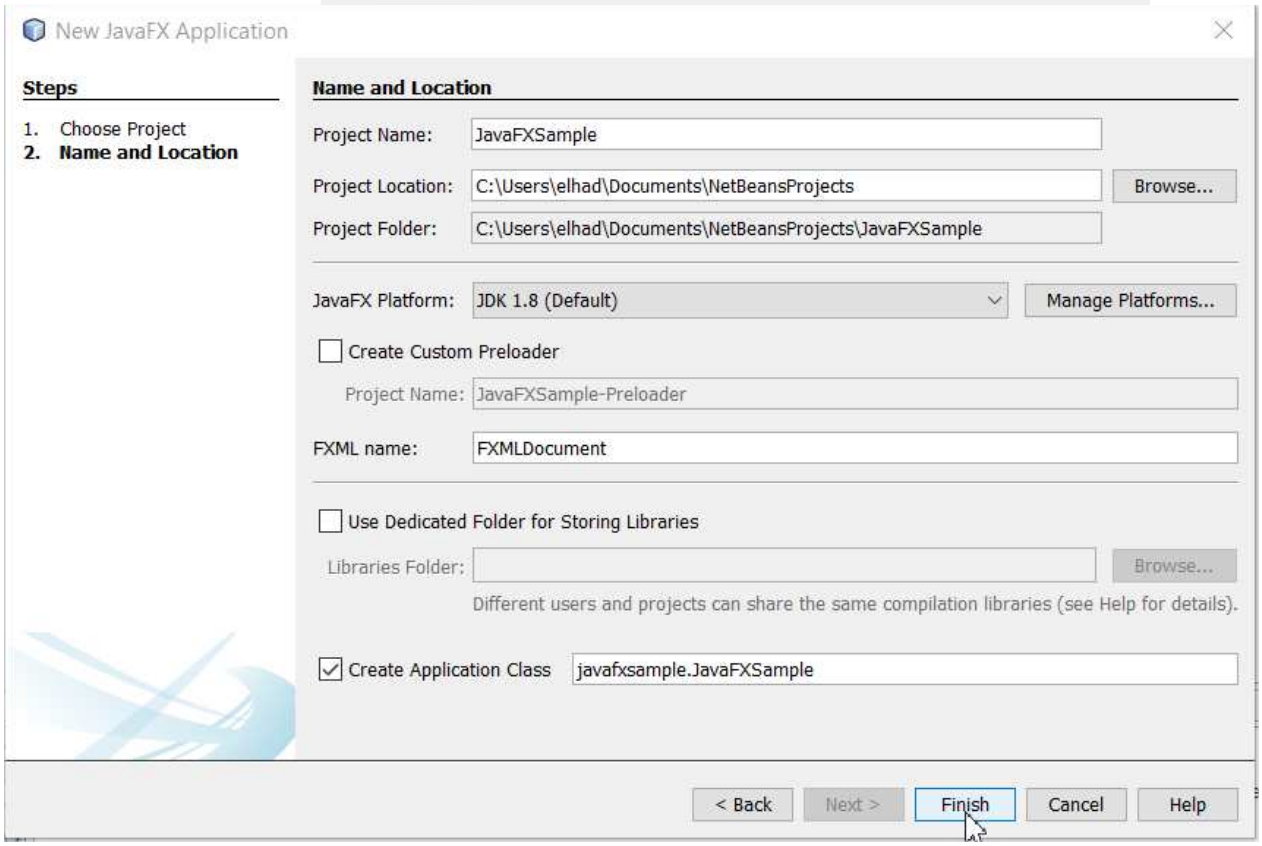




VII) Créer une application Java Fx avec Scène Builder

Créez votre première application Java Fx avec Scène Builder.





L'IDE NetBeans crée les fichiers **JavaFXSample.java**, **FXMLDocumentController.java** et **FXMLDocument.fxml**.

JavaFXSample.java : Ce fichier contient le code Java de démarrage requis pour une application FXML.
FXMLDocument.fxml : C'est le fichier source FXML dans lequel nous allons définir l'interface utilisateur.
FXMLDocumentController.java : C'est le fichier contrôleur pour gérer les différentes évènements (entrées souris, clavier etc...).

Le fichier **FXMLDocument.fxml** aura pour contenu par défaut :

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <?import java.lang.*?>
4   <?import java.util.*?>
5   <?import javafx.scene.*?>
6   <?import javafx.scene.control.*?>
7   <?import javafx.scene.layout.*?>
8
9   <AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://javafx.com/fxml/1"
10  fx:controller="javafxsample.FXMLDocumentController">
11      <children>
12          <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction" fx:id="button" />
13          <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
14      </children>
15  </AnchorPane>
```

En version copiable :

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="javafxsample.FXMLDocumentController">
  <children>
    <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction" fx:id="button" />
    <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
  </children>
</AnchorPane>
```

Ligne 10 : Définition la classe contrôleur **FXMLDocumentController.java**. C'est cette classe qui va gérer les évènements de Java Fx.

Ligne 12 : Définition d'un composant de type Bouton (javafx.scene.control.Button) avec pour identifiant de binding **button**. On définit aussi un évènement lors du clic sur le bouton avec pour identifiant de binding **handleButtonAction**. Plus tard **button et handleButtonAction** seront définis

dans la classe contrôleur **FXMLDocumentController.java**. C'est ce qu'on appelle le Binding c'est-à-dire associer des composants graphiques à une Classe.

Ligne 13 : Définition d'un composant de type Label (javafx.scene.control.Label) avec comme identifiant de Binding **label**. Cet identifiant sera défini dans la classe contrôleur **FXMLDocumentController.java**.

Le fichier **FXMLDocumentController.java** aura pour contenu par défaut :

```java
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

/**
 *
 * @author elhad
 */
public class FXMLDocumentController implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

En version copiable :

```
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

/**
 *
 * @author elhad
 */
public class FXMLDocumentController implements Initializable {

   @FXML
   private Label label;

   @FXML
   private void handleButtonAction(ActionEvent event) {
      System.out.println("You clicked me!");
      label.setText("Hello World!");
```

```
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

Ligne 17 : Définition du composant de type Label avec le nom d'attribut **label** (voir le fichier **FXMLDocument.fxml**). Le fichier **FXMLDocument.fxml** se chargera d'instancier **label**.
Ligne 20 : Définition de l'évènement **handleButtonAction** pour clic sur le bouton **button** (voir le fichier **FXMLDocument.fxml**). Le fichier **FXMLDocument.fxml** instanciera le bouton **button**.

Le fichier **JavaFXSample.java** aura pour contenu par défaut :

```java
package javafxsample;

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

/**
 *
 * @author elhad
 */
public class JavaFXSample extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

En version copiable :

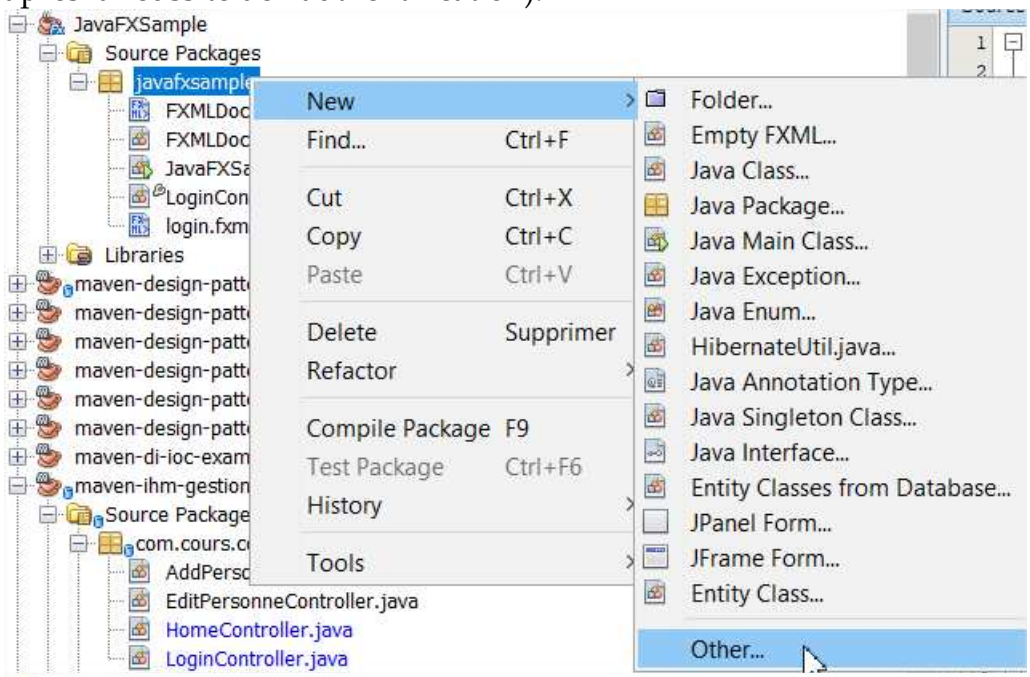
```
package javafxsample;

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

/**
 *
 * @author elhad
 */
public class JavaFXSample extends Application {

   @Override
   public void start(Stage stage) throws Exception {
      Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
      Scene scene = new Scene(root);
      stage.setScene(scene);
```

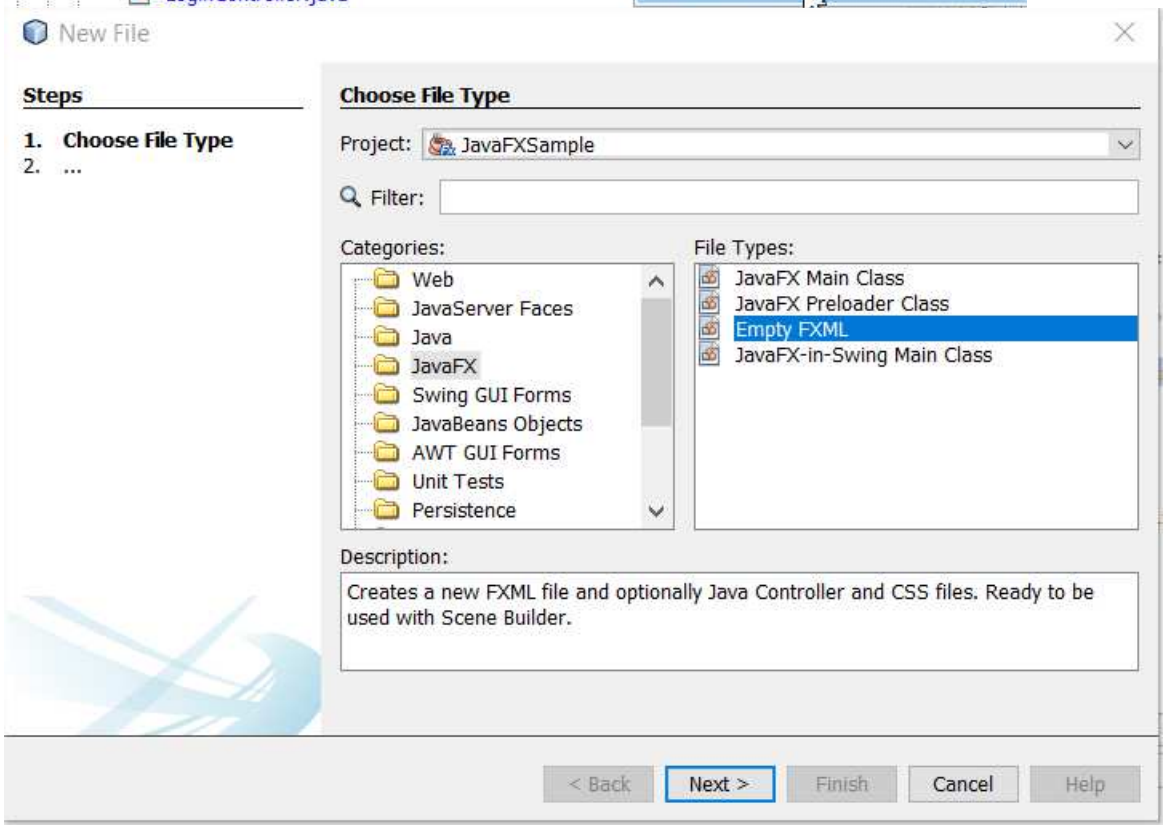
```
        stage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

}
```

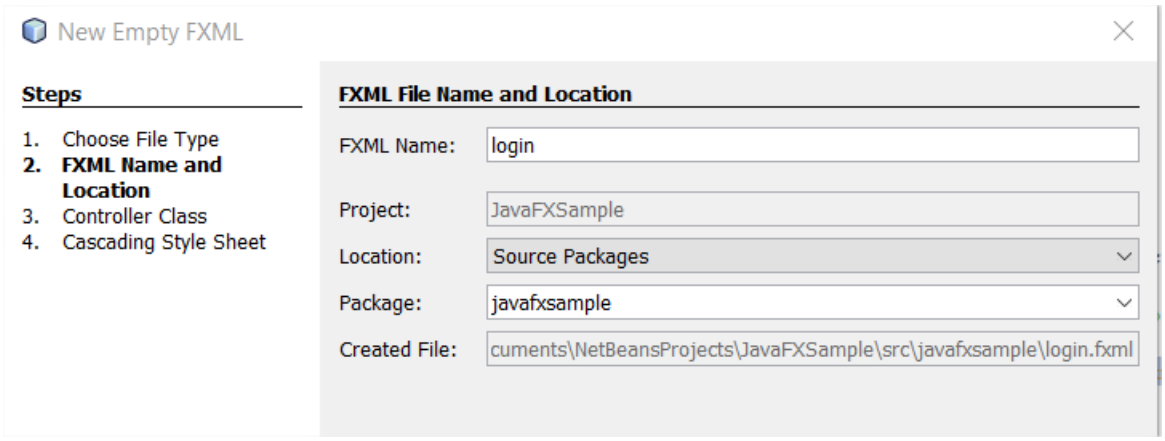
Ligne 17 : Création d'un parent scène à partir **FXMLDocument.fxml**.
Ligne 18 : Affectation du parent scène à la scène principale de l'application.
Ligne 19 : affectation de la « Scene » au « Stage » **stage**.
Ligne 20 : affichage de la fenêtre **stage**.

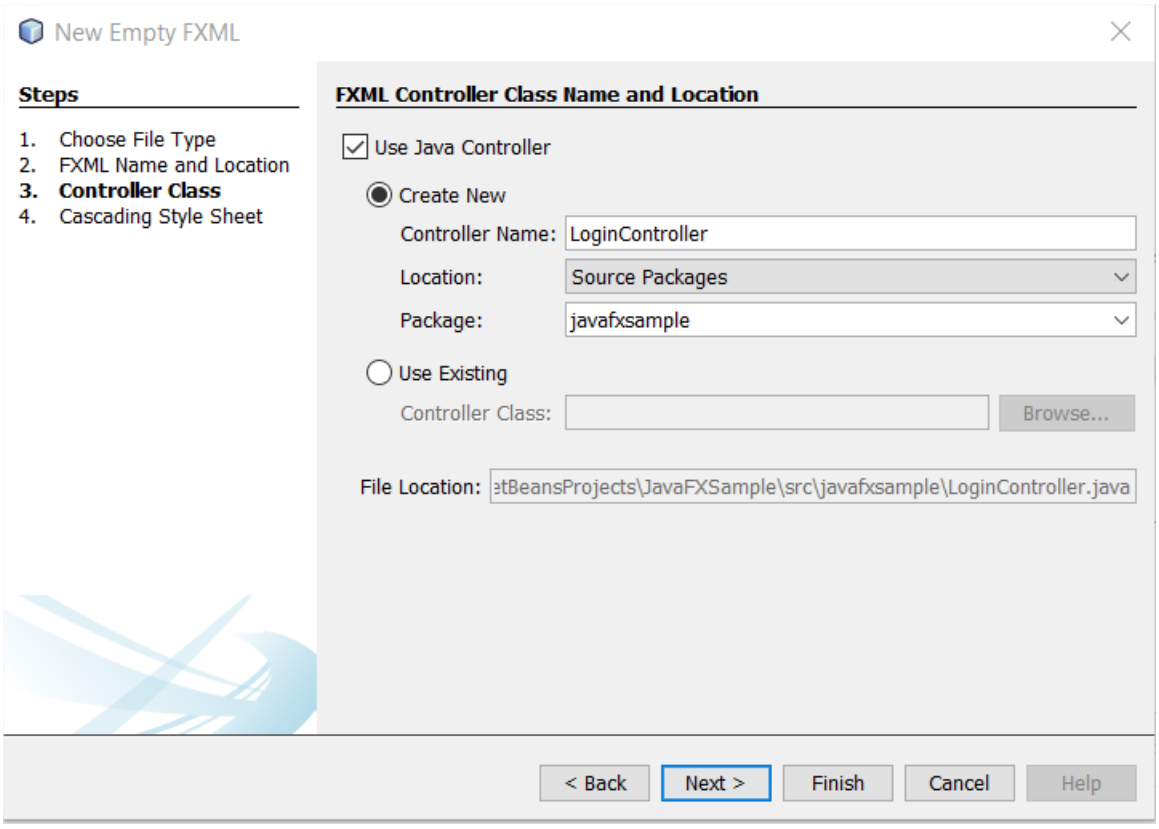
VIII) Créer une application Java Fx Multi Page avec Scène Builder

Nous allons créer une application avec deux pages : une page de connexion et une page d'acceuil. Créer les fichiers **login.fxml** (va servir de page de connexion) et **home.fxml** (sera la page d'acceuil après la reussite de l'authentification).



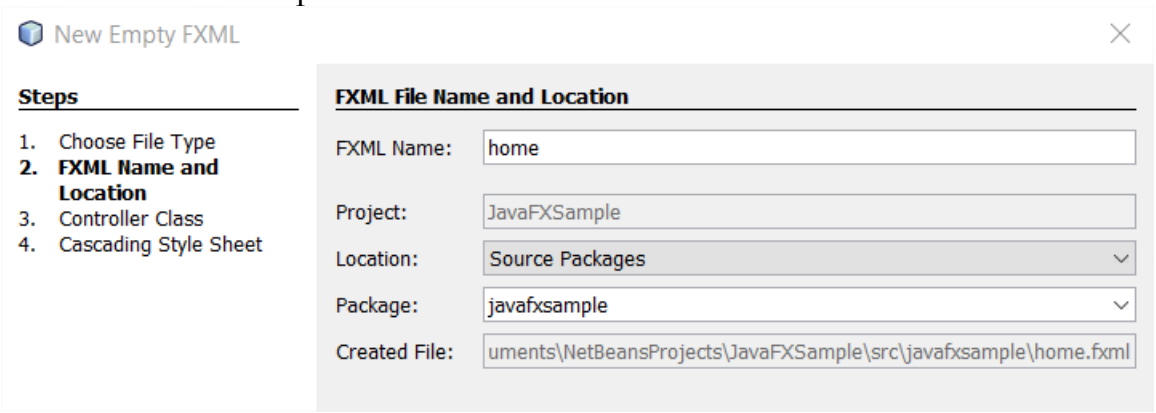


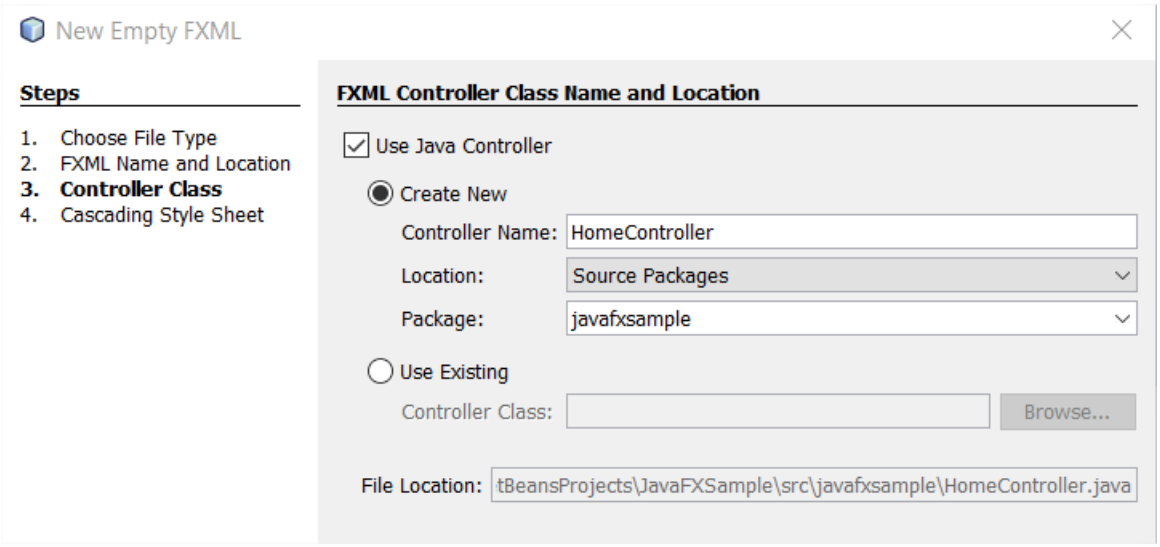




Cliquer sur finish pour finaliser.

Faire la même chose pour **home.fxml**.





login.fxml aura pour contenu par défaut :

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="javafxsample.LoginController">
</AnchorPane>
```

LoginController aura pour contenu par défaut :

```
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;

public class LoginController implements Initializable {

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

home.fxml aura pour contenu par défaut :

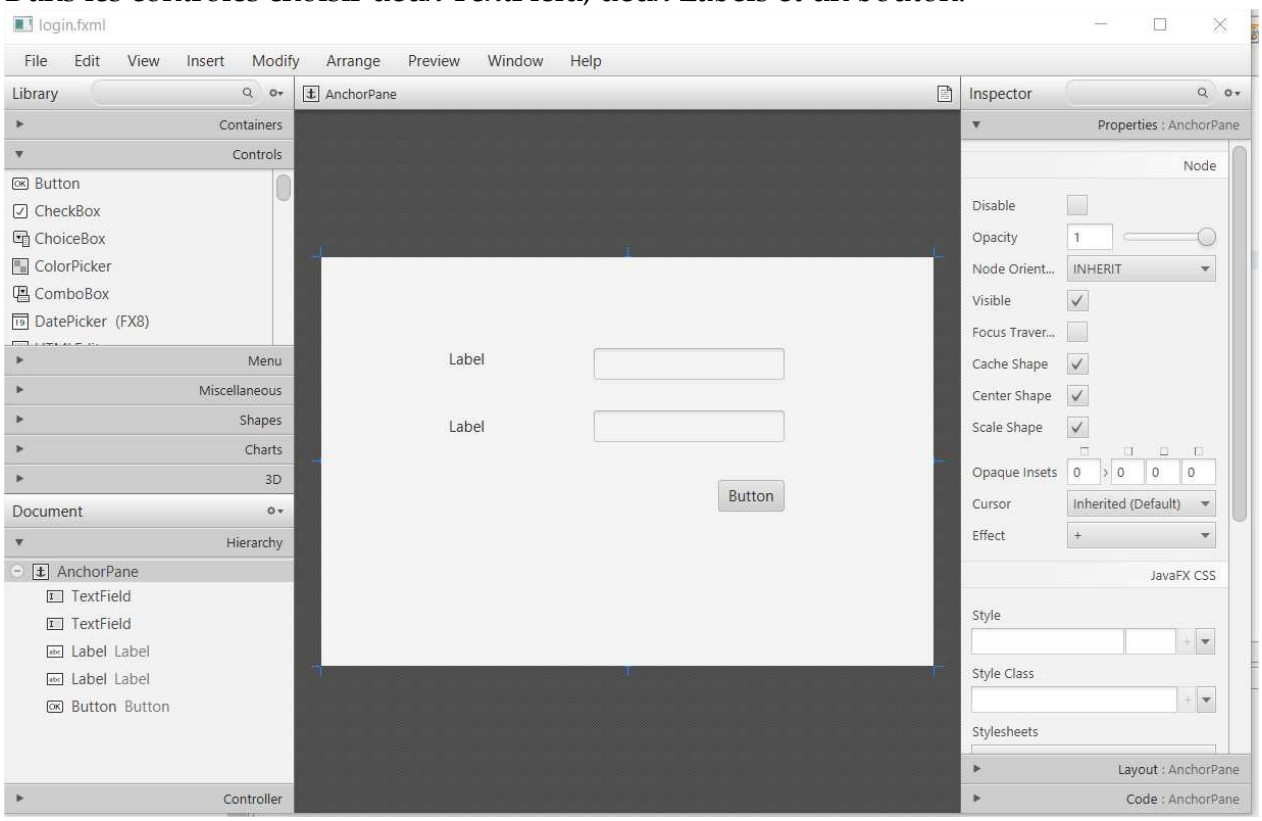
```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="javafxsample.HomeController">
</AnchorPane>
```

HomeController aura pour contenu par défaut :

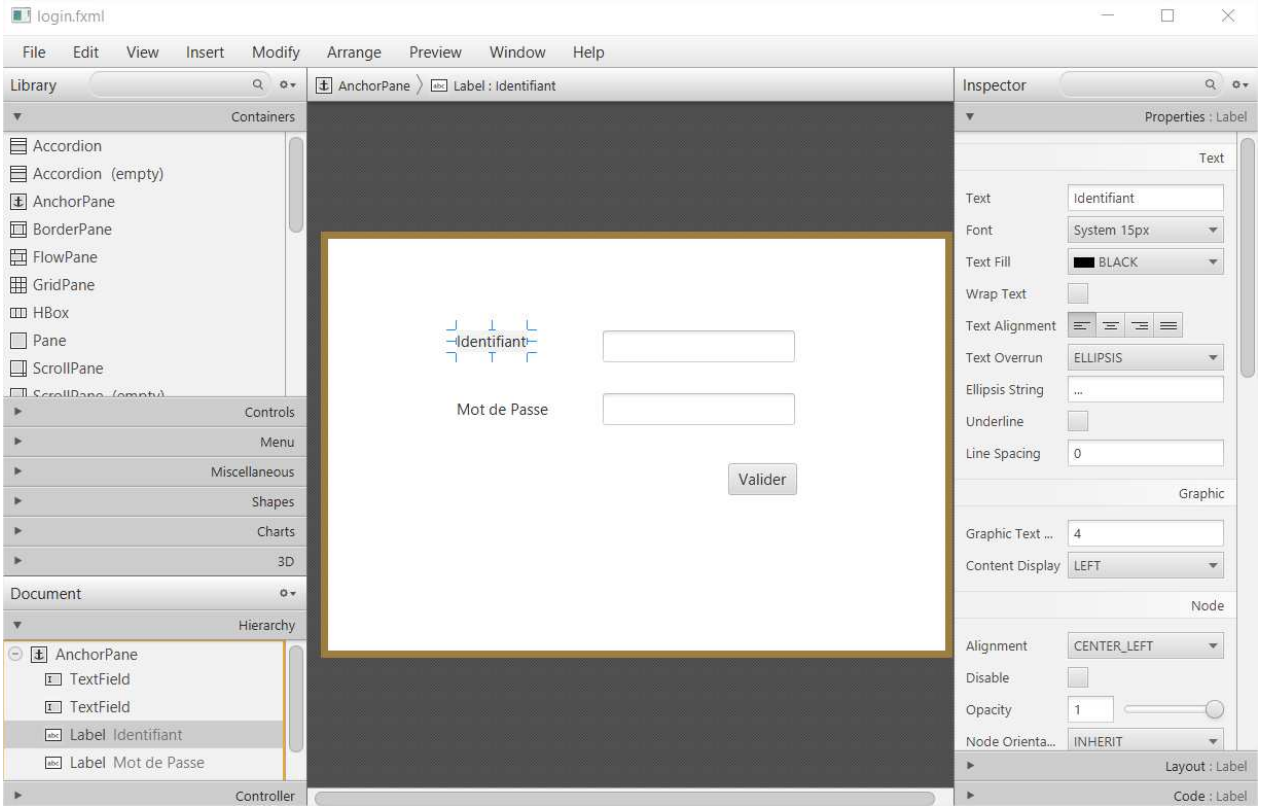
```
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;

public class HomeController implements Initializable {

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

Modifions maintenant le fichier **login.fxml** avec la scène builder :

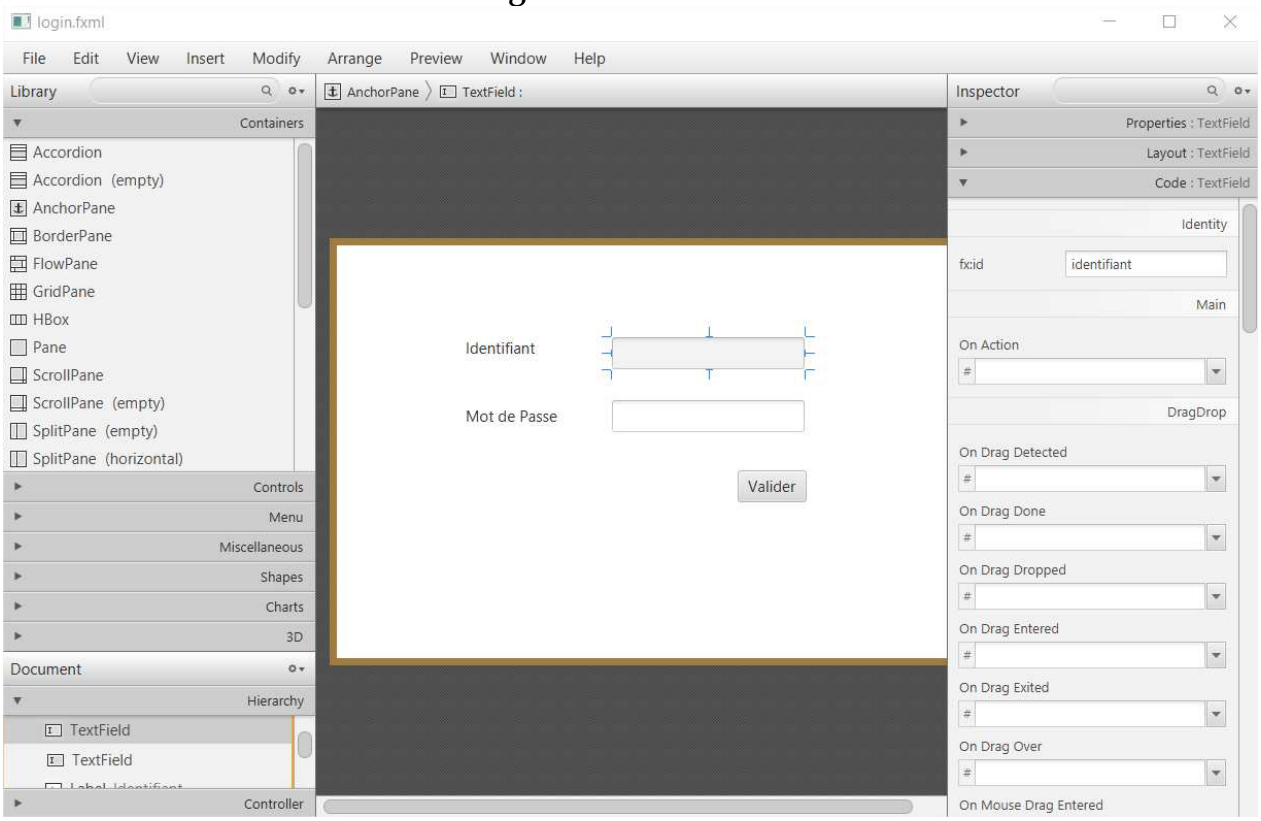
Dans les contrôles choisir deux TextField, deux Labels et un bouton.



Modifier les attributs Text pour mettre les bon textes.

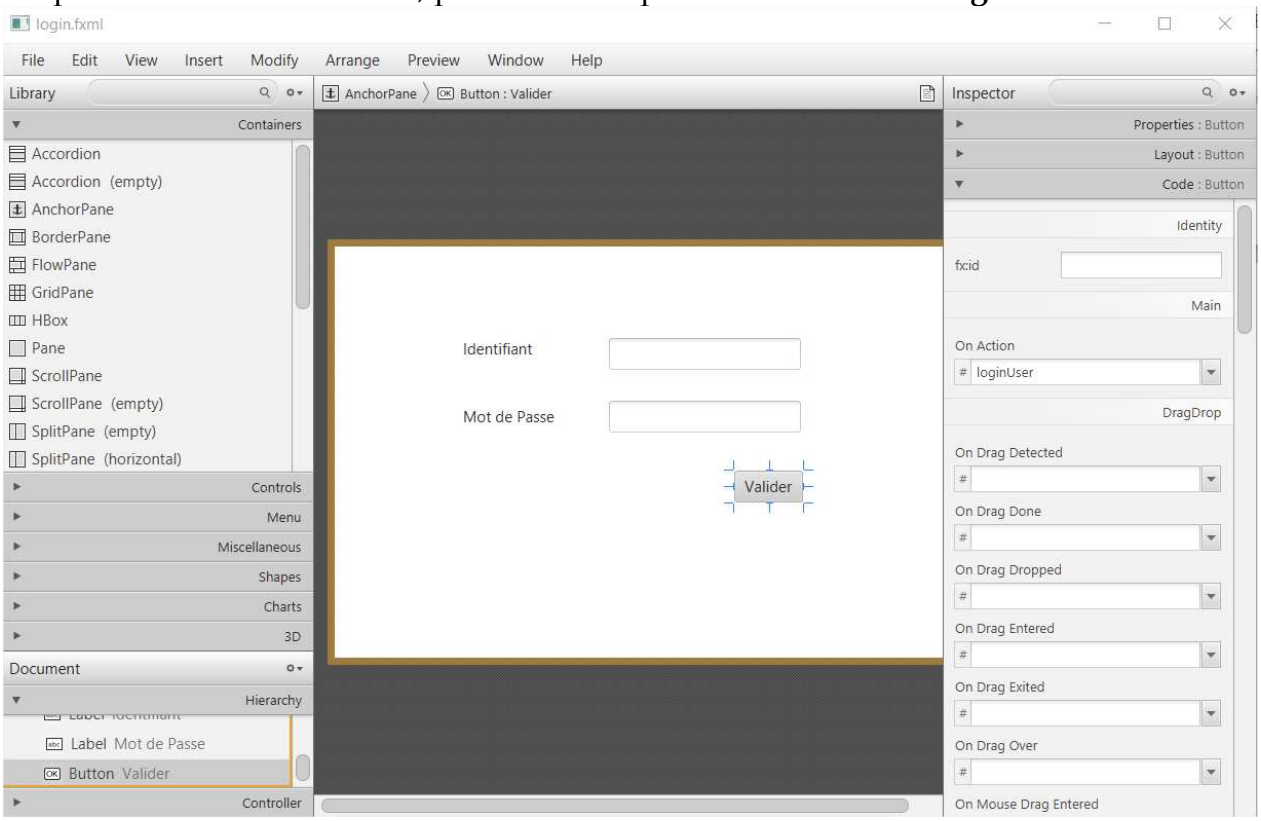


On peut effectuer des binding de composants Java Fx avec les classes controleur pour cela cliquer sur le composant, puis sur code :
On pourra par exemple mettre dans le champ **fx :id** la valeur **identifiant**, nous retrouverons plus tard cet identifiant dans le fichier **login.fxml**.



On peut aussi ajouter des actions aux boutons par exemple soit l'action **loginUser** qui nous permettrera de réaliser l'authentification de l'utilisateur.

Cliquer sur le bouton Valider, puis sur Code puis mettre la valeur **loginUser** dans **On Action**.



Editer le fichier **login.fxml** pour voir le résultat de vos modifications :

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="
    <children>
        <TextField fx:id="identifiant" layoutX="267.0" layoutY="89.0" />
        <TextField fx:id="motPasse" layoutX="267.0" layoutY="150.0" />
        <Label layoutX="125.0" layoutY="89.0" text="Identifiant" />
        <Label layoutX="125.0" layoutY="155.0" text="Mot de Passe" />
        <Button layoutX="389.0" layoutY="218.0" mnemonicParsing="false" onAction="#loginUser" text="Valider" />
    </children>
</AnchorPane>
```

En version copiable :

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="javafxsample.LoginController">
  <children>
    <TextField fx:id="identifiant" layoutX="267.0" layoutY="89.0" />
    <TextField fx:id="motPasse" layoutX="267.0" layoutY="150.0" />
    <Label layoutX="125.0" layoutY="89.0" text="Identifiant" />
    <Label layoutX="125.0" layoutY="155.0" text="Mot de Passe" />
    <Button layoutX="389.0" layoutY="218.0" mnemonicParsing="false" onAction="#loginUser" text="Valider" />
  </children>
</AnchorPane>
```

Vous aurez une erreur dans votre IDE : **Handler method not found**

Ajoutons dans **LoginController** l'action **loginUser** et les composants Java Fx **identifiant** et **motPasse**.

```java
package javafxsample;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.TextField;

public class LoginController implements Initializable {

    @FXML
    private TextField identifiant;
    @FXML
    private TextField motPasse;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    public void loginUser(ActionEvent event) throws IOException {

    }
}
```

En version copiable :

```
package javafxsample;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.TextField;

public class LoginController implements Initializable {

    @FXML
    private TextField identifiant;
    @FXML
    private TextField motPasse;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    public void loginUser(ActionEvent event) throws IOException {

    }
}
```

Ligne 14 : association (binding) du composant graphique avec comme id **identifiant**.
Ligne 16 : association du composant graphique avec comme id **motPasse**.
Ligne 26 : Définition de la méthode **loginUser** (initialement défini dans le fichier **login.fxml**) dans le contrôleur.

Modifier la classe **JavaFXSample** pour charger la page **login.fxml** au démarrage de l'application :

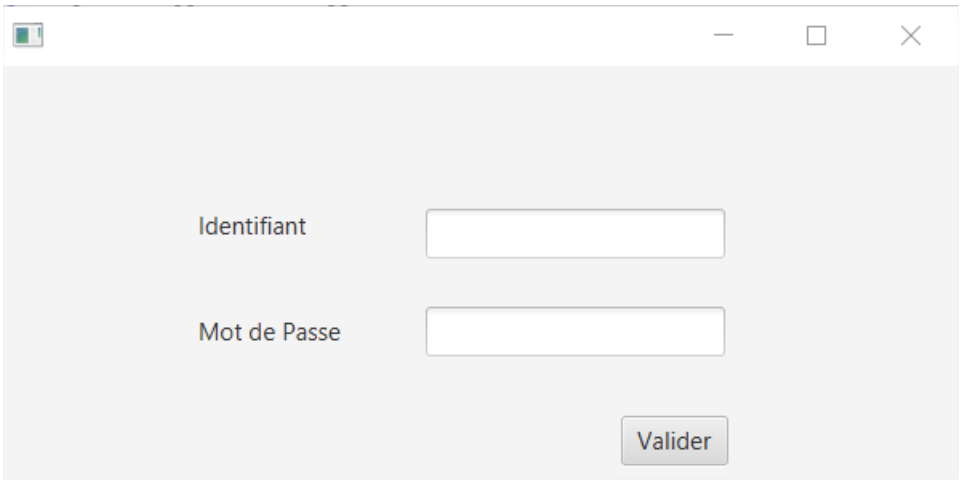
```java
package javafxsample;

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

public class JavaFXSample extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        loadLoginPage(stage);
    }

    public void loadLoginPage(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public void loadSamplePage(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

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

```

En version copiable :

```
package javafxsample;

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

public class JavaFXSample extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        loadLoginPage(stage);
    }

    public void loadLoginPage(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public void loadSamplePage(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

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

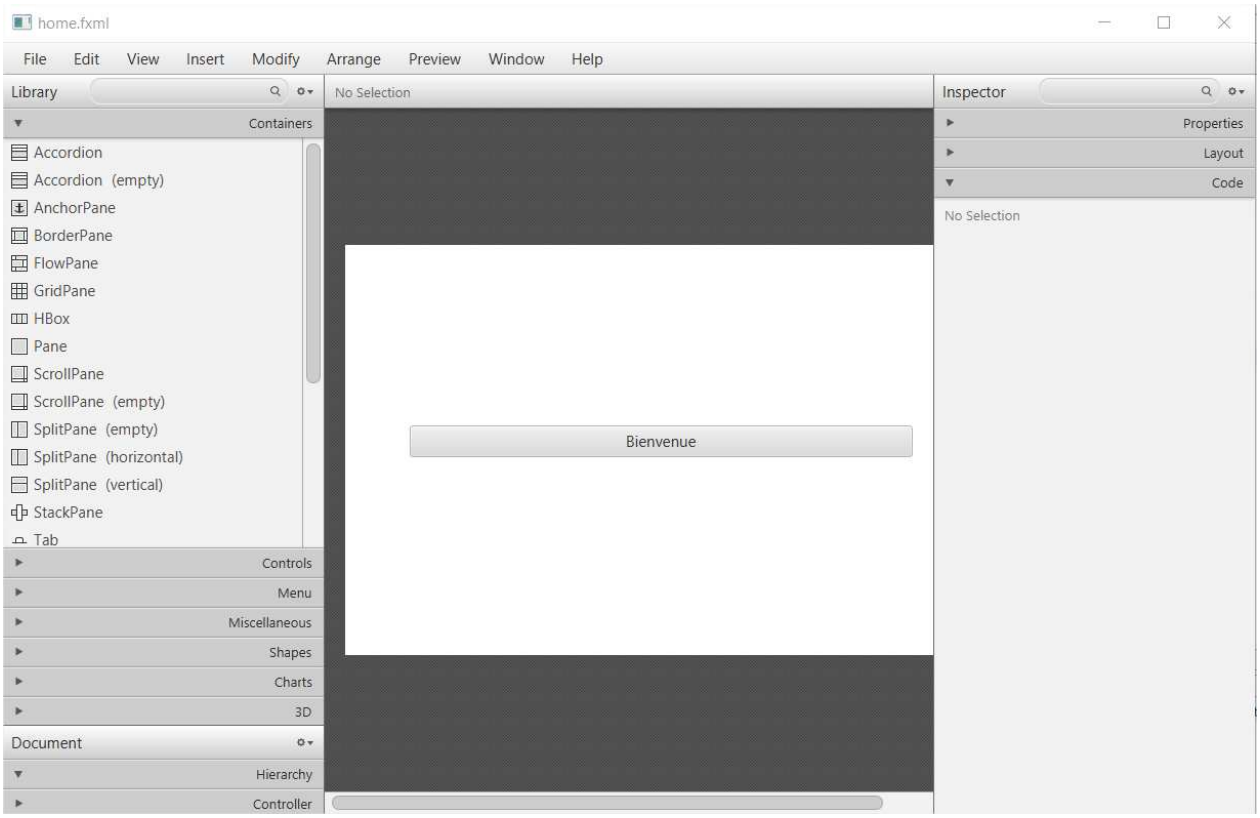
Ligne 13 : appel de **loadLoginPage** pour l'affichage de la page de login.

On obtient donc comme résultat (après un Clean And Build):



On peut par exemple afficher la page **home.fxml** lorsque dans la page **login.fxml** on tape comme identifiant **admin** et comme mot de passe **admin**.

Ajoutons un bouton de bienvenue avec l'id **welcomeButton** dans **home.fxml** :



home.fxml devient :

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="javafxsample.HomeController">
  <children>
    <Button fx:id="welcomeButton" layoutX="63.0" layoutY="176.0" mnemonicParsing="false" prefHeight="31.0"
prefWidth="491.0" text="Bienvenue" />
  </children>
</AnchorPane>
```

Ajouter ensuite **welcomeButton** dans HomeController :

```
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;

public class HomeController implements Initializable {

    @FXML
    private Button welcomeButton;

    public HomeController() {
        super();
    }

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
    }

    public Button getWelcomeButton() {
        return welcomeButton;
    }
}
```

En version copiable :

```
package javafxsample;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;

public class HomeController implements Initializable {

    @FXML
    private Button welcomeButton;

    public HomeController() {
        super();
    }

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
    }

    public Button getWelcomeButton() {
        return welcomeButton;
    }
}
```

La classe **LoginController** devient :

```
15
16  public class LoginController implements Initializable {
17
18      @FXML
19      private TextField identifiant;
20      @FXML
21      private TextField motPasse;
22
23      /**
24       * Initializes the controller class.
25       */
26      @Override
27      public void initialize(URL url, ResourceBundle rb) {
28          // TODO
29      }
30
31      public void loginUser(ActionEvent event) throws IOException {
32          if ("admin".equals(identifiant.getText()) && "admin".equals(motPasse.getText())) {
33              FXMLLoader loader = new FXMLLoader(getClass().getResource("home.fxml"));
34              Parent homeParent = loader.load();
35              HomeController homeController = loader.getController();
36              homeController.getWelcomeButton().setText("Bienvenue Mr " + identifiant.getText() + " dans votre p
37              Scene scene = new Scene(homeParent);
38              Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
39              currentStage.setScene(scene);
40              currentStage.show();
41          }
42      }
43  }
```

En version copiable :

```
package javafxsample;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class LoginController implements Initializable {

   @FXML
   private TextField identifiant;
   @FXML
   private TextField motPasse;

   /**
```

```
    * Initializes the controller class.
    */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    public void loginUser(ActionEvent event) throws IOException {
        if ("admin".equals(identifiant.getText()) && "admin".equals(motPasse.getText())) {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("home.fxml"));
            Parent homeParent = loader.load();
            HomeController homeController = loader.getController();
            homeController.getWelcomeButton().setText("Bienvenue Mr " + identifiant.getText() + " dans votre page
d'acceuil");
            Scene scene = new Scene(homeParent);
            Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
            currentStage.setScene(scene);
            currentStage.show();
        }
    }
}
```

On obtient à l'exécution :