eBook Gratuit

APPRENEZ javafx

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#javafx

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: javafx

It is an unofficial and free javafx ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official javafx.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec javafx

Remarques

JavaFX est une plate-forme logicielle permettant de créer et de fournir des applications de bureau, ainsi que des applications Internet riches (RIA) pouvant fonctionner sur une grande variété de périphériques. JavaFX est destiné à remplacer Swing en tant que bibliothèque graphique standard pour Java SE.

Le service informatique permet aux développeurs de concevoir, créer, tester, déboguer et déployer des applications client enrichi.

L'apparence des applications JavaFX peut être personnalisée à l'aide de feuilles de style CSS (Cascading Style Sheets) pour le style (voir JavaFX: CSS) et (F) Les fichiers XML peuvent être utilisés pour créer ou développer une application (voir FXML et Contrôleurs). . Scene Builder est un éditeur visuel permettant la création de fichiers fxml pour une interface utilisateur sans écrire de code.

Versions

Version	Date de sortie
JavaFX 2	2011-10-10
JavaFX 8	2014-03-18

Examples

Installation ou configuration

> Les API JavaFX sont disponibles en tant que fonctionnalité entièrement intégrée à l'environnement d'exécution Java (JRE) et au kit de développement Java (JDK). Le JDK étant disponible pour toutes les principales plates-formes de bureau (Windows, Mac OS X et Linux), les applications JavaFX compilées pour JDK 7 et versions ultérieures s'exécutent également sur toutes les principales plates-formes de bureau. La prise en charge des plates-formes ARM est également disponible avec JavaFX 8. JDK for ARM inclut les composants base, graphics et controls de JavaFX.

Pour installer JavaFX, installez la version choisie de l'environnement Java Runtime et du kit de développement Java .

Les fonctionnalités offertes par JavaFX incluent:

1. API Java.
2. FXML et Scene Builder.

3. WebView
4. Interopérabilité Swing.
5. Contrôles d'interface utilisateur et CSS intégrés.
6. Thème Modène.
7. Caractéristiques graphiques 3D.
8. API de toile.
9. API d'impression.
10. Prise en charge de texte enrichi.
11. Prise en charge multitouch
12. Support Hi-DPI.
13. Pipeline graphique accéléré par le matériel.
14. Moteur multimédia haute performance.
15. Modèle de déploiement d'application autonome.

Programme Hello World

Le code suivant crée une interface utilisateur simple contenant un seul `Button` qui imprime une `String` à la console en cliquant.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        // create a button with specified text
        Button button = new Button("Say 'Hello World'");

        // set a handler that is executed when the user activates the button
        // e.g. by clicking it or pressing enter while it's focused
        button.setOnAction(e -> {
           //Open information dialog that says hello
           Alert alert = new Alert(AlertType.INFORMATION, "Hello World!?");
           alert.showAndWait();
        });

        // the root of the scene shown in the main window
        StackPane root = new StackPane();

        // add button as child of the root
        root.getChildren().add(button);

        // create a scene specifying the root and the size
        Scene scene = new Scene(root, 500, 300);

        // add scene to the stage
        primaryStage.setScene(scene);

        // make the stage visible
```

```
        primaryStage.show();
    }

    public static void main(String[] args) {
        // launch the HelloWorld application.

        // Since this method is a member of the HelloWorld class the first
        // parameter is not required
        Application.launch(HelloWorld.class, args);
    }

}
```

La classe d' `Application` est le point d'entrée de chaque application JavaFX. Une seule `Application` peut être lancée et ceci en utilisant

```
Application.launch(HelloWorld.class, args);
```

Cela crée une instance de la classe `Application` passée en paramètre et démarre la plate-forme JavaFX.

Ce qui suit est important pour le programmeur ici:

1. Le premier `launch` crée une nouvelle instance de la classe `Application` (`HelloWorld` dans ce cas). La classe `Application` donc besoin d'un constructeur no-arg.
2. `init()` est appelé sur l'instance d' `Application` créée. Dans ce cas, l'implémentation par défaut de l' `Application` ne fait rien.
3. `start` est appelé pour l'instance `Appication` et le `Stage` principal (= window) est transmis à la méthode. Cette méthode est appelée automatiquement sur le thread d'application JavaFX (thread thread).
4. L'application s'exécute jusqu'à ce que la plateforme détermine qu'il est temps d'arrêter. Cela se fait lorsque la dernière fenêtre est fermée dans ce cas.
5. La méthode d' `stop` est appelée sur l'instance d' `Application` . Dans ce cas, l'implémentation à partir de l' `Application` ne fait rien. Cette méthode est appelée automatiquement sur le thread d'application JavaFX (thread thread).

Dans la méthode de `start` , le graphe de scène est construit. Dans ce cas, il contient 2 `Node` : un `Button` et un `StackPane` .

Le `Button` représente un bouton dans l'interface utilisateur et le `StackPane` est un conteneur pour le `Button` qui détermine son emplacement.

Une `Scene` est créée pour afficher ces `Node` . Enfin, la `Scene` est ajoutée à la `Stage` qui est la fenêtre qui montre l’intégralité de l’interface utilisateur.

Lire Démarrer avec javafx en ligne: https://riptutorial.com/fr/javafx/topic/887/demarrer-avec-javafx

Chapitre 2: Animation

Examples

Animation d'une propriété avec ligne de temps

```
Button button = new Button("I'm here...");

Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80))
);
t.setAutoReverse(true);
t.setCycleCount(Timeline.INDEFINITE);
t.play();
```

Le moyen le plus simple et le plus souple d'utiliser l'animation dans JavaFX est la classe `Timeline` . Une chronologie fonctionne en utilisant `KeyFrame` comme points connus dans l'animation. Dans ce cas, il sait qu'au début (`0 seconds`) le `translateXProperty` doit être à zéro et à la fin (`2 seconds`) que la propriété doit être `80` . Vous pouvez également faire d'autres choses comme définir l'animation pour inverser et combien de fois il devrait fonctionner.

Les timelines peuvent animer plusieurs propriétés en même temps:

```
Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(1), new KeyValue(button.translateYProperty(), 10)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80)),
        new KeyFrame(Duration.seconds(3), new KeyValue(button.translateYProperty(), 90))
);                                                              //   ^ notice X vs Y
```

Cette animation amènera la propriété `Y` de `0` (valeur initiale de la propriété) à `10` secondes, et se terminera à `90` secondes trois secondes. Notez que lorsque l'animation commence au-delà, `Y` retourne à zéro, même si ce n'est pas la première valeur du scénario.

Lire Animation en ligne: https://riptutorial.com/fr/javafx/topic/5166/animation

Chapitre 3: Bouton

Examples

Ajouter un écouteur d'action

Les boutons déclenchent des événements d'action lorsqu'ils sont activés (par exemple, cliqué, une reliure du bouton est pressée, ...).

```
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

Si vous utilisez Java 8+, vous pouvez utiliser lambdas pour les auditeurs d'action.

```
button.setOnAction((ActionEvent a) -> System.out.println("Hello, World!"));
// or
button.setOnAction(a -> System.out.println("Hello, World!"));
```

Ajouter un graphique à un bouton

Les boutons peuvent avoir un graphique. `graphic` peut être n'importe quel noeud JavaFX, comme un `ProgressBar`

```
button.setGraphic(new ProgressBar(-1));
```

Une `ImageView`

```
button.setGraphic(new ImageView("images/icon.png"));
```

Ou même un autre bouton

```
button.setGraphic(new Button("Nested button"));
```

Créer un bouton

La création d'un `Button` est simple:

```
Button sampleButton = new Button();
```

Cela créera un nouveau `Button` sans texte ni graphique à l'intérieur.

Si vous souhaitez créer un `Button` avec un texte, utilisez simplement le constructeur qui prend un

paramètre `String` comme paramètre (ce qui définit la `textProperty` du `Button`):

```
Button sampleButton = new Button("Click Me!");
```

Si vous souhaitez créer un `Button` avec un graphique à l'intérieur ou tout autre `Node` , utilisez ce constructeur:

```
Button sampleButton = new Button("I have an icon", new ImageView(new Image("icon.png")));
```

Boutons par défaut et Annuler

`Button` API de `Button` permet d'attribuer facilement des raccourcis clavier communs aux boutons sans avoir besoin d'accéder à la liste des accélérateurs affectée à `Scene` ou d'écouter explicitement les événements clés. À savoir, deux méthodes de commodité sont fournies: `setDefaultButton` et `setCancelButton` :

- `setDefaultButton` vous `setDefaultButton` sur `true` , le `Button` se déclenche chaque fois qu'il reçoit un événement `KeyCode.ENTER` .
- `setCancelButton` vous `setCancelButton` sur `true` , le `Button` se déclenche chaque fois qu'il reçoit un événement `KeyCode.ESCAPE` .

L'exemple suivant crée une `Scene` avec deux boutons activés lorsque vous appuyez sur les touches Entrée ou Echap, qu'elles soient ou non mises au point.

```
FlowPane root = new FlowPane();

Button okButton = new Button("OK");
okButton.setDefaultButton(true);
okButton.setOnAction(e -> {
    System.out.println("OK clicked.");
});

Button cancelButton = new Button("Cancel");
cancelButton.setCancelButton(true);
cancelButton.setOnAction(e -> {
    System.out.println("Cancel clicked.");
});

root.getChildren().addAll(okButton, cancelButton);
Scene scene = new Scene(root);
```

Le code ci-dessus ne fonctionnera pas si ces éléments `KeyEvents` sont utilisés par un `Node` parent:

```
scene.setOnKeyPressed(e -> {
    e.consume();
});
```

Lire Bouton en ligne: https://riptutorial.com/fr/javafx/topic/5162/bouton

Chapitre 4: Bouton radio

Examples

Création de boutons radio

Les boutons radio permettent à l'utilisateur de choisir un élément parmi ceux donnés. Il existe deux manières de déclarer un `RadioButton` avec un texte à côté de lui. Soit en utilisant le constructeur par défaut `RadioButton()` et en définissant le texte avec la `setText(String)` ou en utilisant l'autre constructeur `RadioButton(String)` .

```
RadioButton radioButton1 = new RadioButton();
radioButton1.setText("Select me!");
RadioButton radioButton2= new RadioButton("Or me!");
```

Comme `RadioButton` est une extension de `Labeled` il peut également y avoir une `Image` spécifiée pour `RadioButton` . Après avoir créé le `RadioButton` avec l'un des constructeurs, ajoutez simplement la méthode `Image` avec la `setGraphic(ImageView)` comme ici:

```
Image image = new Image("ok.jpg");
RadioButton radioButton = new RadioButton("Agree");
radioButton.setGraphic(new ImageView(image));
```

Utiliser des groupes sur les boutons radio

Un groupe `ToggleGroup` est utilisé pour gérer les `ToggleGroup` `RadioButton` manière à ce qu'un seul groupe soit sélectionné à chaque fois.

Créez un simple `ToggleGroup` comme suit:

```
ToggleGroup group = new ToggleGroup();
```

Après avoir créé un groupe `Togglegroup` il peut être assigné à `RadioButton` en utilisant `setToggleGroup(ToggleGroup)` . Utilisez `setSelected(Boolean)` pour présélectionner l'un des s `RadioButton` .

```
RadioButton radioButton1 = new RadioButton("stackoverlow is awesome! :)");
radioButton1.setToggleGroup(group);
radioButton1.setSelected(true);

RadioButton radioButton2 = new RadioButton("stackoverflow is ok :|");
radioButton2.setToggleGroup(group);

RadioButton radioButton3 = new RadioButton("stackoverflow is useless :(");
radioButton3.setToggleGroup(group);
```

Événements pour les boutons radio

Généralement, lorsque l'un des `ToggleGroup` `RadioButton` d'un groupe `ToggleGroup` est sélectionné, l'application exécute une action. Voici un exemple qui imprime les données utilisateur du `RadioButton` sélectionné qui a été défini avec `setUserData(Object)` .

```
radioButton1.setUserData("awesome")
radioButton2.setUserData("ok");
radioButton3.setUserData("useless");

ToggleGroup group = new ToggleGroup();
group.selectedToggleProperty().addListener((obserableValue, old_toggle, new_toggle) -> {
    if (group.getSelectedToggle() != null) {
        System.out.println("You think that stackoverflow is " +
group.getSelectedToggle().getUserData().toString());
    }
});
```

Demande de focus pour les boutons radio

Disons que le second `RadioButton` sur trois est présélectionné avec `setSelected(Boolean)` , le focus est toujours au premier `RadioButton` par défaut. Pour changer cela, utilisez la méthode `requestFocus()` .

```
radioButton2.setSelected(true);
radioButton2.requestFocus();
```

Lire Bouton radio en ligne: https://riptutorial.com/fr/javafx/topic/5906/bouton-radio

Chapitre 5: Créateur de scène

Introduction

JavaFX Scene Builder est un outil de présentation visuelle qui permet aux utilisateurs de concevoir rapidement des interfaces utilisateur d'application JavaFX, sans codage. Il est utilisé pour générer des fichiers FXML.

Remarques

JavaFX Scene Builder est un outil de présentation visuelle qui permet aux utilisateurs de concevoir rapidement des interfaces utilisateur d'application JavaFX, sans codage. Les utilisateurs peuvent faire glisser et déposer les composants de l'interface utilisateur dans une zone de travail, modifier leurs propriétés, appliquer des feuilles de style et le code FXML de la mise en page qu'ils créent est automatiquement généré en arrière-plan. Le résultat est un fichier FXML qui peut ensuite être combiné avec un projet Java en liant l'interface utilisateur à la logique de l'application.

Dans une perspective MVC (Model View Controller):

- Le fichier FXML, contenant la description de l'interface utilisateur, est la vue.
- Le contrôleur est une classe Java, implémentant éventuellement la classe Initializable, qui est déclarée en tant que contrôleur pour le fichier FXML.
- Le modèle se compose d'objets de domaine, définis du côté Java, pouvant être connectés à la vue via le contrôleur.

Installation du créateur de scène

1. Téléchargez la version la plus récente de Scene Builder sur le site Web de Gluon, en sélectionnant le programme d'installation correspondant à votre plate-forme ou à l'exécutable.

2. Une fois le programme d'installation téléchargé, double-cliquez pour installer Scene Builder sur votre système. Un JRE mis à jour est inclus.

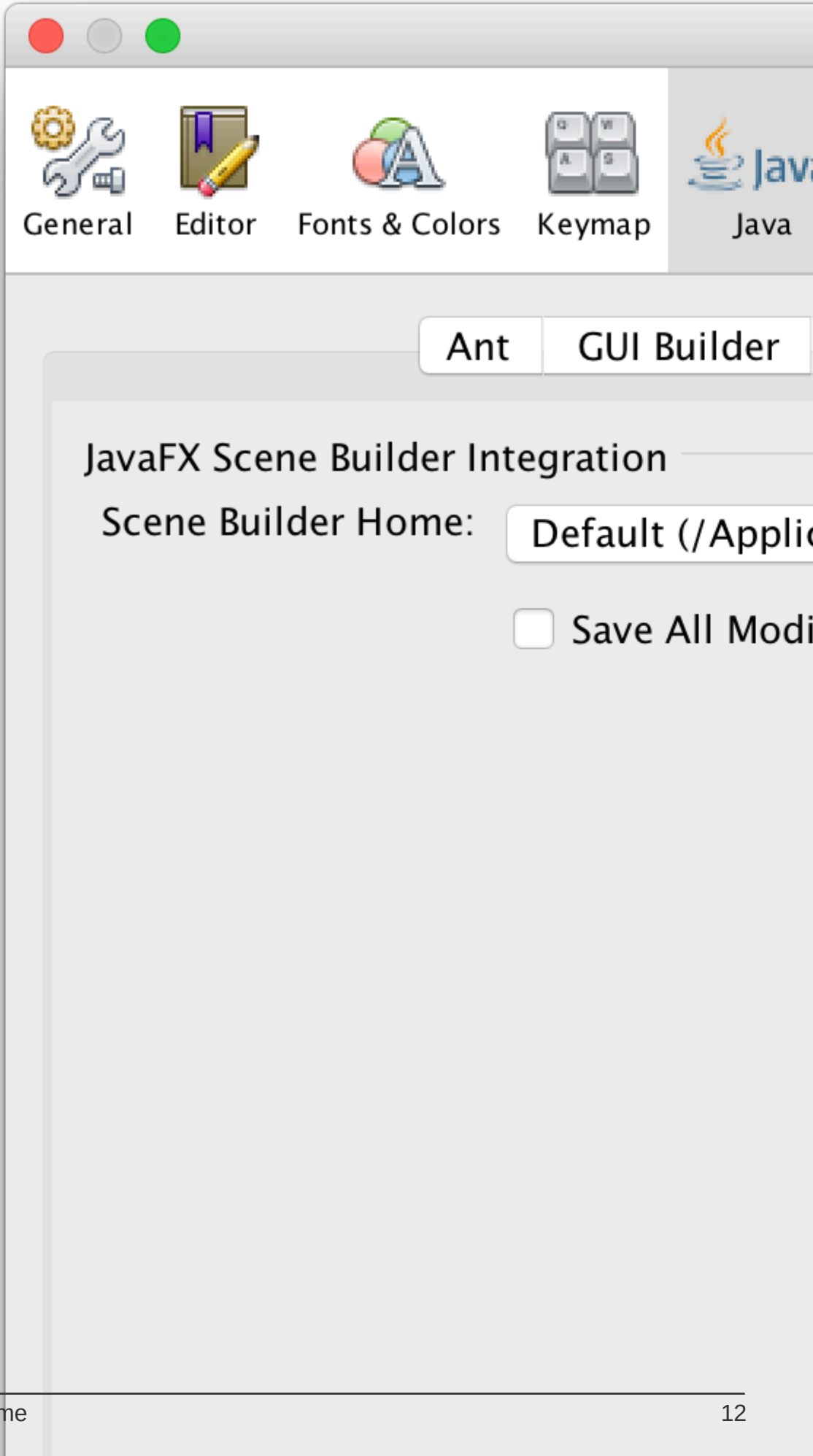
3. Double-cliquez sur l'icône Scene Builder pour l'exécuter en tant qu'application autonome.

4. Intégration IDE

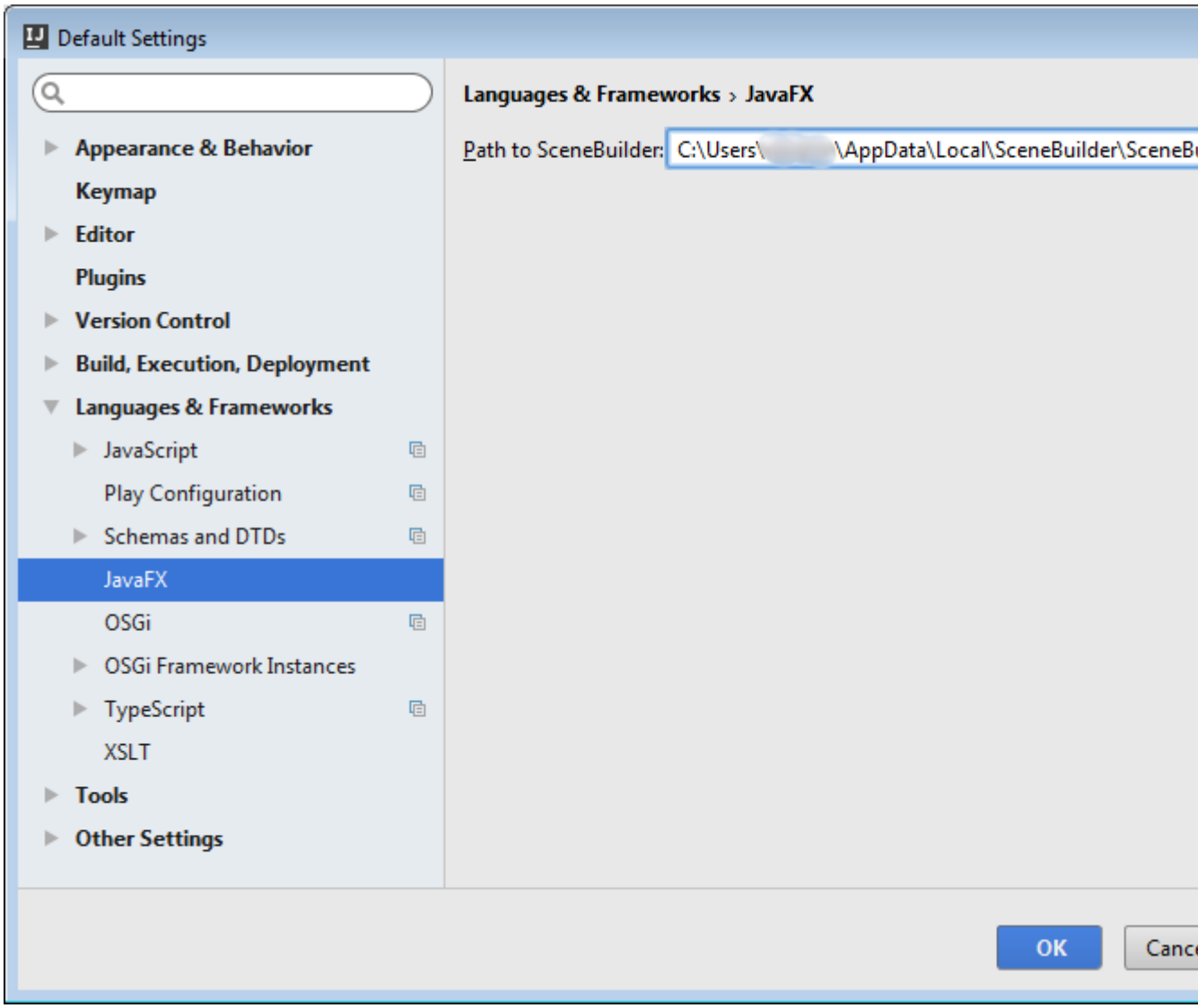
   Bien que Scene Builder soit une application autonome, il produit des fichiers FXML intégrés à un projet Java SE. Lors de la création de ce projet sur un IDE, il est pratique d'inclure un lien vers le chemin du Générateur de scènes, afin que les fichiers FXML puissent être édités.

   - NetBeans: Sous Windows, accédez à NetBeans-> Tools-> Options-> Java-> JavaFX. Sous Mac OS X, accédez à NetBeans-> Preferences-> Java-> JavaFX. Indiquez le

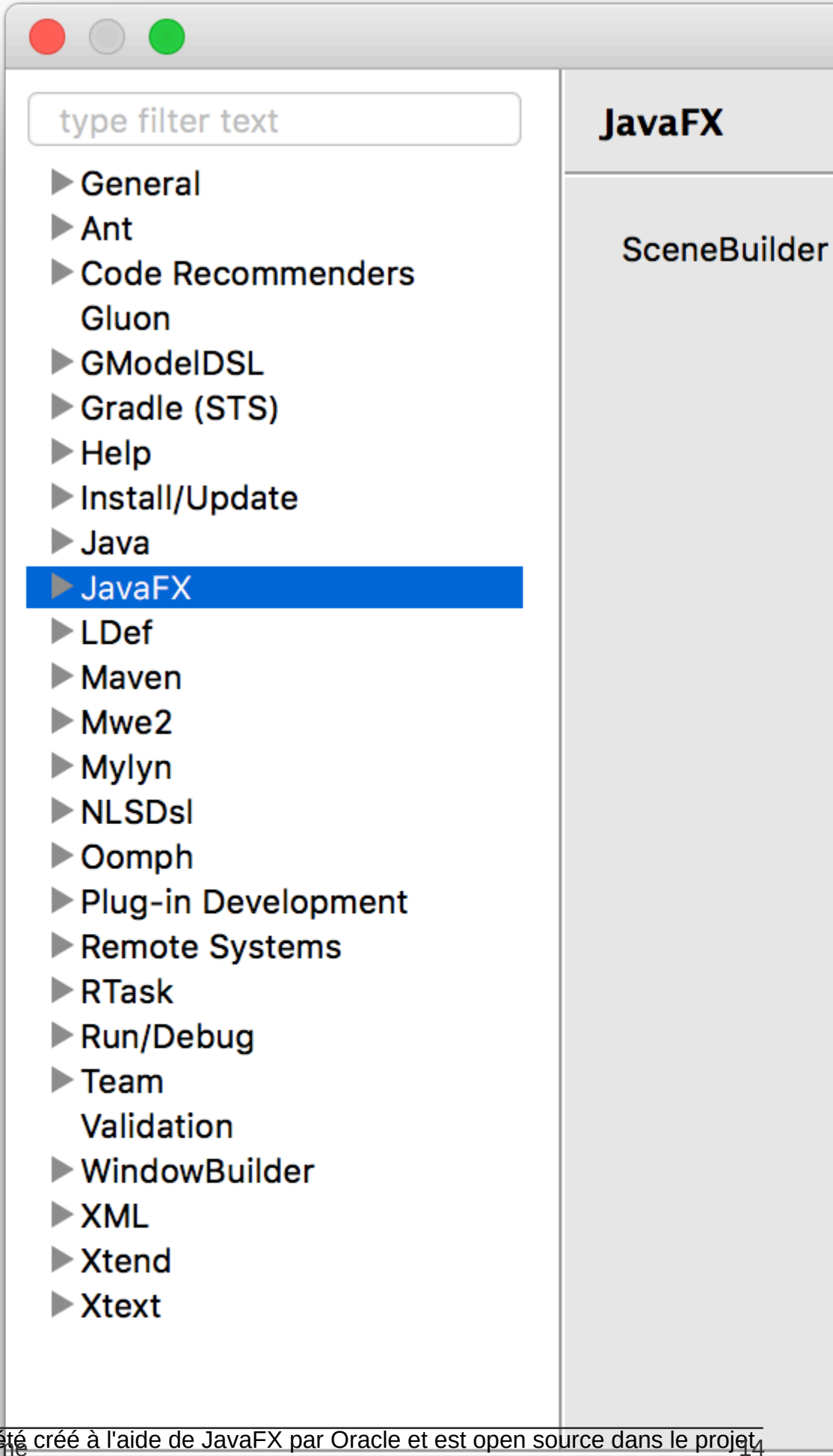
chemin d'accès à la maison Scene Builder.


General
Editor
Fonts & Colors
Keymap
Java
Ant
GUI Builder
JavaFX Scene Builder Integration
Scene Builder Home:
Default (/Applic
Save All Modi

- IntelliJ: Sous Windows, accédez à IntelliJ-> Paramètres-> Langues et cadres-> JavaFX. Sous Mac OS X, accédez à IntelliJ-> Preferences-> Languages & Frameworks-> JavaFX. Indiquez le chemin d'accès à la maison Scene Builder.



- Eclipse: Sous Windows, accédez à Eclipse-> Window-> Preferences-> JavaFX. Sous Mac OS X, accédez à Eclipse-> Preferences-> JavaFX. Indiquez le chemin d'accès à la maison Scene Builder.



Le projet Scene Builder a été créé à l'aide de JavaFX par Oracle et est open source dans le projet

Un peu d'histoire

Oracle a fourni des

fichiers binaires, jusqu'à Scene Builder v 2.0, incluant uniquement des fonctionnalités JavaFX avant la sortie de Java SE 8u40. De nouvelles fonctionnalités telles que les contrôles `Spinner` ne sont donc pas incluses.

Gluon a repris la distribution des versions binaires, et un Scene Builder 8+ mis à jour peut être téléchargé pour chaque plate-forme à partir d' ici .

Il inclut les dernières modifications de JavaFX, ainsi que les améliorations récentes et les corrections de bogues.

Le projet open source peut être trouvé ici où des problèmes, des demandes de fonctionnalités et des requêtes d'extraction peuvent être créés.

Les anciens binaires Oracle peuvent toujours être téléchargés ici .

Des tutoriels

Vous trouverez des tutoriels sur Scene Builder ici:

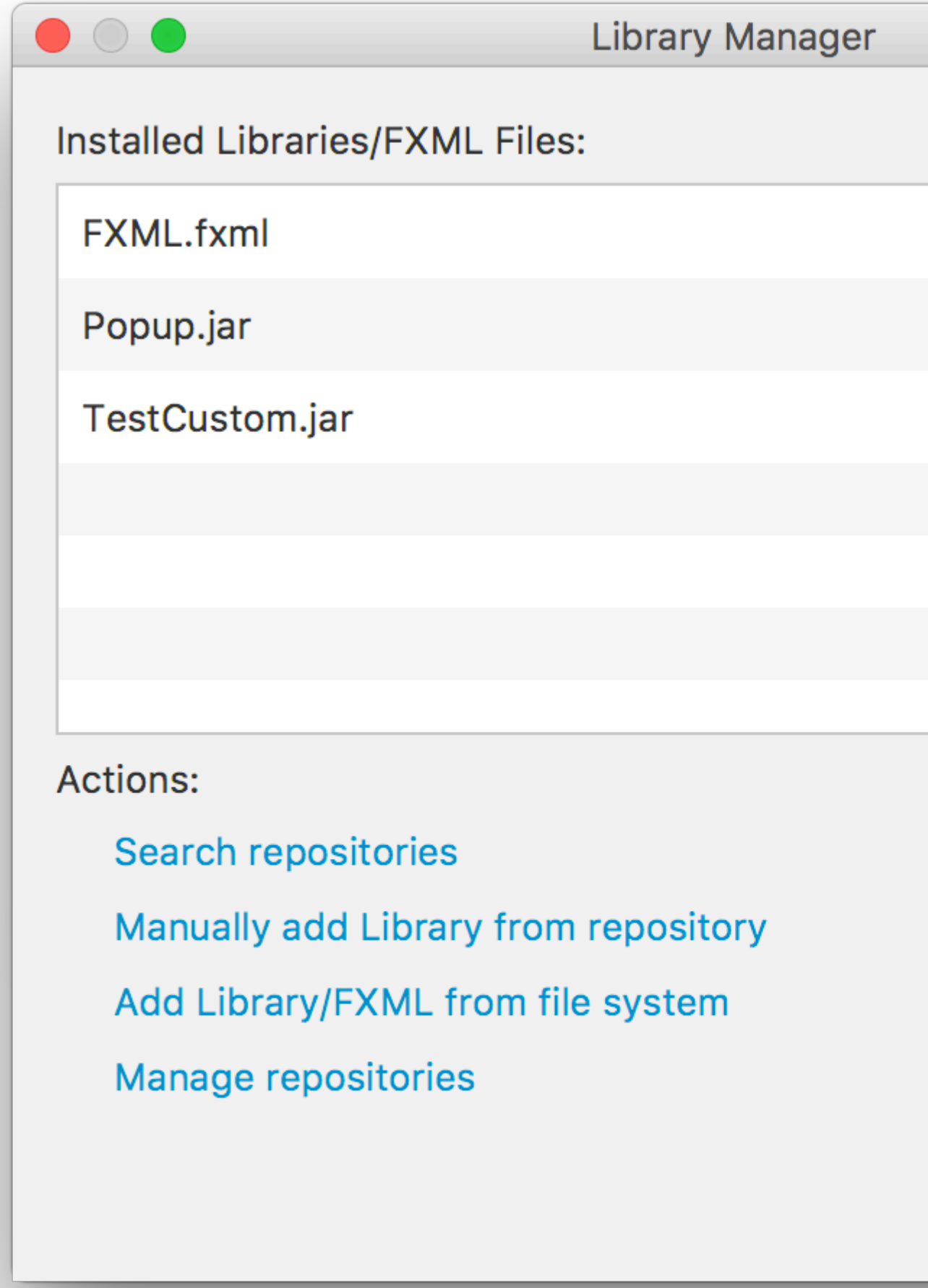
- Tutoriel Oracle Scene Builder 2.0

Les tutoriels FXML peuvent être trouvés ici.

- Oracle FXML tutoriel

Contrôles personnalisés

Gluon a entièrement documenté la nouvelle fonctionnalité qui permet d'importer des fichiers jar tiers avec des contrôles personnalisés, à l'aide du gestionnaire de bibliothèque (disponible depuis Scene Builder 8.2.0).



Tag le. SceneBuilder

`basicFXML.fxml` jar / classpath, comme spécifié par `FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))` .

Lors du chargement de `basicFXML.fxml` , le chargeur trouvera le nom de la classe du contrôleur, comme spécifié par `fx:controller="org.stackoverflow.BasicFXMLController"` dans le fichier FXML.

Ensuite, le chargeur créera une instance de cette classe, dans laquelle il tentera d'injecter tous les objets ayant un `fx:id` dans le fichier FXML et qui seront marqués avec l'annotation `@FXML` dans la classe du contrôleur.

Dans cet exemple, le FXMLLoader créera le libellé basé sur `<Label ... fx:id="label"/>` et injectera l'occurrence de l'étiquette dans le `@FXML private Label label;` .

Enfin, lorsque l'ensemble du fichier FXML a été chargé, le FXMLLoader appelle la méthode d' `initialize` du contrôleur et le code qui enregistre un gestionnaire d'action avec le bouton est exécuté.

Édition

Bien que le fichier FXML puisse être modifié dans l'EDI, il n'est pas recommandé, car l'EDI ne fournit qu'une vérification de base de la syntaxe et une auto-complétion, mais pas de guidage visuel.

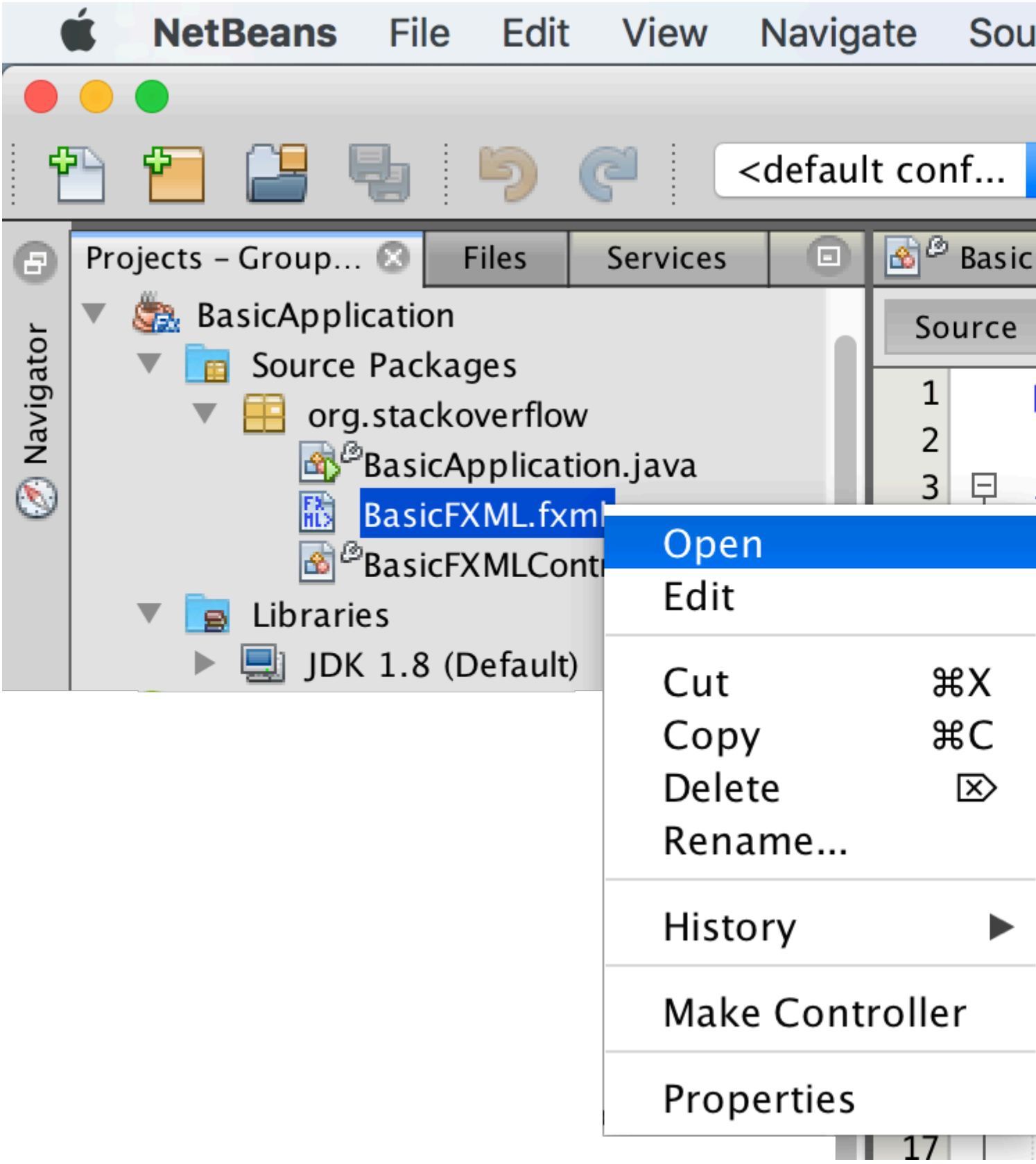
La meilleure approche consiste à ouvrir le fichier FXML avec Scene Builder, où toutes les modifications seront enregistrées dans le fichier.

Scene Builder peut être lancé pour ouvrir le fichier:

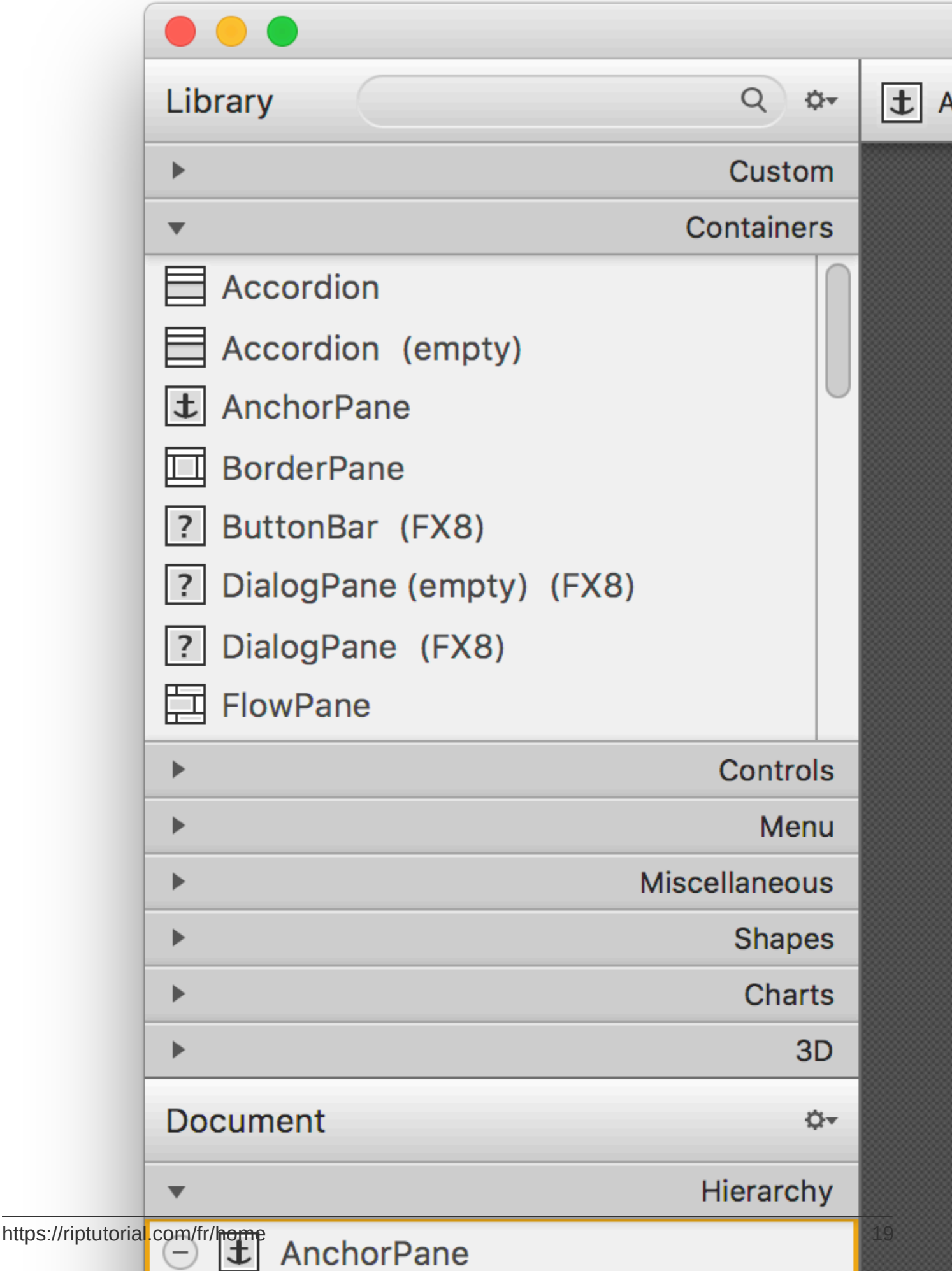
Ou le fichier peut être ouvert avec Scene Builder directement depuis l'EDI:

- Dans NetBeans, dans l'onglet Projet, double-cliquez sur le fichier ou cliquez avec le bouton droit de la souris et sélectionnez `Open` .
- Dans IntelliJ, dans l'onglet Projet, cliquez avec le bouton droit sur le fichier et sélectionnez `Open In Scene Builder` .

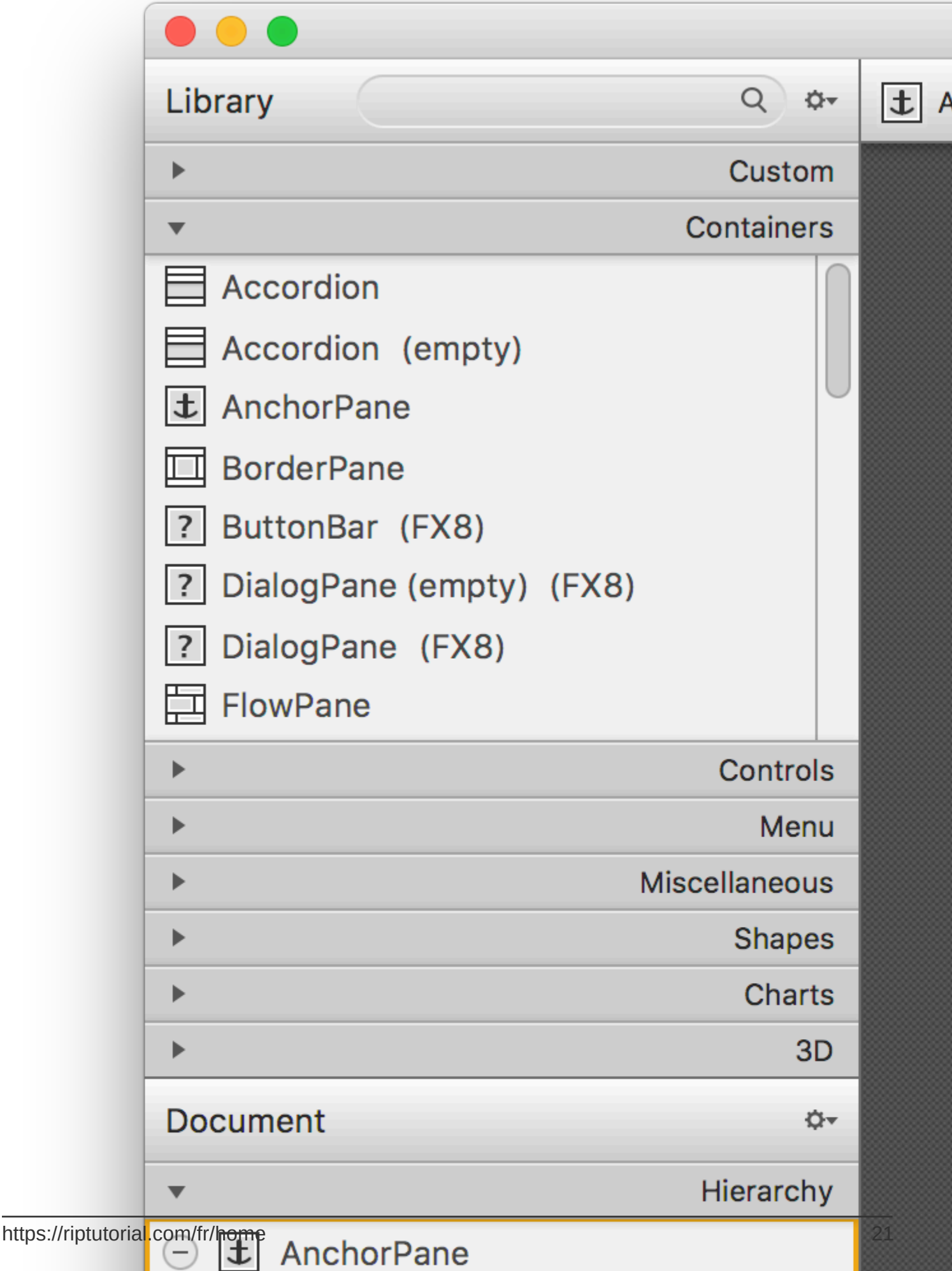
- Dans Eclipse, dans l'onglet Projet, cliquez avec le bouton droit sur le fichier et sélectionnez `Open with Scene Builder`.



Si Scene Builder est correctement installé et que son chemin d'accès est ajouté à l'EDI (voir Remarques ci-dessous), il ouvre le fichier:


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

. Il peut être défini dans le volet `Code` :


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

Chapitre 6: CSS

Syntaxe

- NodeClass / * sélecteur par classe de noeud * /
- .someclass / * selector par classe * /
- #someId / * selector par id * /
- [sélecteur1]> [sélecteur2] / * sélecteur pour un enfant direct d'un noeud correspondant au sélecteur1 correspondant à selector2 * /
- [sélecteur1] [sélecteur2] / * sélecteur pour un descendant d'un nœud correspondant à selector1 qui correspond à selector2 * /

Examples

Utilisation de CSS pour le style

Le CSS peut être appliqué à plusieurs endroits:

- inline (`Node.setStyle`)
- dans une feuille de style
  - à une `Scene`
    - comme feuille de style de l'agent utilisateur (non démontré ici)
    - comme feuille de style "normale" pour la `Scene`
  - à un `Node`

Cela permet de modifier les propriétés de style des `Nodes` . L'exemple suivant illustre ceci:

Classe d'application

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Region;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class StyledApplication extends Application {

    @Override
    public void start(Stage primaryStage) {

        Region region1 = new Region();
        Region region2 = new Region();
        Region region3 = new Region();
        Region region4 = new Region();
        Region region5 = new Region();
        Region region6 = new Region();

        // inline style
```

```
        region1.setStyle("-fx-background-color: yellow;");

        // set id for styling
        region2.setId("region2");

        // add class for styling
        region2.getStyleClass().add("round");
        region3.getStyleClass().add("round");

        HBox hBox = new HBox(region3, region4, region5);

        VBox vBox = new VBox(region1, hBox, region2, region6);

        Scene scene = new Scene(vBox, 500, 500);

        // add stylesheet for root
        scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());

        // add stylesheet for hBox
        hBox.getStylesheets().add(getClass().getResource("inlinestyle.css").toExternalForm());

        scene.setFill(Color.BLACK);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

inlinestyle.css

```
* {
    -fx-opacity: 0.5;
}

HBox {
    -fx-spacing: 10;
}

Region {
    -fx-background-color: white;
}
```

style.css

```
Region {
    width: 50;
    height: 70;

    -fx-min-width: width;
    -fx-max-width: width;

    -fx-min-height: height;
    -fx-max-height: height;
```

```
    -fx-background-color: red;
}

VBox {
    -fx-spacing: 30;
    -fx-padding: 20;
}

#region2 {
    -fx-background-color: blue;
}
```

Extension du rectangle en ajoutant de nouvelles propriétés stylisées

JavaFX 8

L'exemple suivant montre comment ajouter des propriétés personnalisées pouvant être stylisées de CSS en `Node` personnalisé.

Ici, 2 `DoubleProperty` s sont ajoutés à la classe `Rectangle` pour permettre de définir la `width` et la `height` partir de CSS.

Le code CSS suivant peut être utilisé pour styliser le noeud personnalisé:

```
 StyleableRectangle {
    -fx-fill: brown;
    -fx-width: 20;
    -fx-height: 25;
    -fx-cursor: hand;
}
```

Noeud personnalisé

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import javafx.beans.property.DoubleProperty;
import javafx.css.CssMetaData;
import javafx.css.SimpleStyleableDoubleProperty;
import javafx.css.StyleConverter;
import javafx.css.Styleable;
import javafx.css.StyleableDoubleProperty;
import javafx.css.StyleableProperty;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Rectangle;

public class StyleableRectangle extends Rectangle {

    // declaration of the new properties
    private final StyleableDoubleProperty styleableWidth = new
SimpleStyleableDoubleProperty(WIDTH_META_DATA, this, "styleableWidth");
    private final StyleableDoubleProperty styleableHeight = new
SimpleStyleableDoubleProperty(HEIGHT_META_DATA, this, "styleableHeight");
```

```
    public StyleableRectangle() {
        bind();
    }

    public StyleableRectangle(double width, double height) {
        super(width, height);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double width, double height, Paint fill) {
        super(width, height, fill);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double x, double y, double width, double height) {
        super(x, y, width, height);
        initStyleableSize();
        bind();
    }

    private void initStyleableSize() {
        styleableWidth.set(getWidth());
        styleableHeight.set(getHeight());
    }

    private final static List<CssMetaData<? extends Styleable, ?>> CLASS_CSS_META_DATA;

    // css metadata for the width property
    // specify property name as -fx-width and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> WIDTH_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-width", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            // property can be set iff the property is not bound
            return !styleable.styleableWidth.isBound();
        }

        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
            // extract the property from the styleable
            return styleable.styleableWidth;
        }
    };

    // css metadata for the height property
    // specify property name as -fx-height and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> HEIGHT_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-height", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            return !styleable.styleableHeight.isBound();
        }

        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
```

```
            return styleable.styleableHeight;
        }
    };

    static {
        // combine already available properties in Rectangle with new properties
        List<CssMetaData<? extends Styleable, ?>> parent = Rectangle.getClassCssMetaData();
        List<CssMetaData<? extends Styleable, ?>> additional = Arrays.asList(HEIGHT_META_DATA,
WIDTH_META_DATA);

        // create arraylist with suitable capacity
        List<CssMetaData<? extends Styleable, ?>> own = new ArrayList(parent.size()+
additional.size());

        // fill list with old and new metadata
        own.addAll(parent);
        own.addAll(additional);

        // make sure the metadata list is not modifiable
        CLASS_CSS_META_DATA = Collections.unmodifiableList(own);
    }

    // make metadata available for extending the class
    public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    // returns a list of the css metadata for the stylable properties of the Node
    @Override
    public List<CssMetaData<? extends Styleable, ?>> getCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    private void bind() {
        this.widthProperty().bind(this.styleableWidth);
        this.heightProperty().bind(this.styleableHeight);
    }


    // -------------------------------------------------------------------------
    // ----------------------- PROPERTY METHODS --------------------------------
    // -------------------------------------------------------------------------

    public final double getStyleableHeight() {
        return this.styleableHeight.get();
    }

    public final void setStyleableHeight(double value) {
        this.styleableHeight.set(value);
    }

    public final DoubleProperty styleableHeightProperty() {
        return this.styleableHeight;
    }

    public final double getStyleableWidth() {
        return this.styleableWidth.get();
    }

    public final void setStyleableWidth(double value) {
        this.styleableWidth.set(value);
```

```
    }

    public final DoubleProperty styleableWidthProperty() {
        return this.styleableWidth;
    }

}
```

Lire CSS en ligne: https://riptutorial.com/fr/javafx/topic/1581/css

Chapitre 7: Dialogues

Remarques

Les dialogues ont été ajoutés dans JavaFX 8 update 40.

Examples

TextInputDialog

`TextInputDialog` permet à l'utilisateur de saisir une seule `String` .

```
TextInputDialog dialog = new TextInputDialog("42");
dialog.setHeaderText("Input your favourite int.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite int: ");

Optional<String> result = dialog.showAndWait();

String s = result.map(r -> {
    try {
        Integer n = Integer.valueOf(r);
        return MessageFormat.format("Nice! I like {0} too!", n);
    } catch (NumberFormatException ex) {
        return MessageFormat.format("Unfortunately \"{0}\" is not a int!", r);
    }
}).orElse("You really don't want to tell me, huh?");

System.out.println(s);
```

ChoiceDialog

`ChoiceDialog` permet à l'utilisateur de choisir un élément dans une liste d'options.

```
List<String> options = new ArrayList<>();
options.add("42");
options.add("9");
options.add("467829");
options.add("Other");

ChoiceDialog<String> dialog = new ChoiceDialog<>(options.get(0), options);
dialog.setHeaderText("Choose your favourite number.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite number:");

Optional<String> choice = dialog.showAndWait();

String s = choice.map(c -> "Other".equals(c) ?
              "Unfortunately your favourite number is not available!"
              : "Nice! I like " + c + " too!")
           .orElse("You really don't want to tell me, huh?");
```

```
System.out.println(s);
```

Alerte

`Alert` est un simple popup qui affiche un ensemble de boutons et obtient un résultat en fonction du bouton sur lequel l'utilisateur a cliqué:

Exemple

Cela permet à l'utilisateur de décider si (s) il veut vraiment fermer l'étape primaire:

```
@Override
public void start(Stage primaryStage) {
    Scene scene = new Scene(new Group(), 100, 100);

    primaryStage.setOnCloseRequest(evt -> {
        // allow user to decide between yes and no
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Do you really want to close
this application?", ButtonType.YES, ButtonType.NO);

        // clicking X also means no
        ButtonType result = alert.showAndWait().orElse(ButtonType.NO);

        if (ButtonType.NO.equals(result)) {
            // consume event i.e. ignore close request
            evt.consume();
        }
    });
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Notez que le texte du bouton est automatiquement ajusté en fonction des `Locale` .

Texte du bouton personnalisé

Le texte affiché dans un bouton peut être personnalisé en créant vous-même une instance `ButtonType` :

```
ButtonType answer = new ButtonType("42");
ButtonType somethingElse = new ButtonType("54");

Alert alert = new Alert(Alert.AlertType.NONE, "What do you get when you multiply six by
nine?", answer, somethingElse);
ButtonType result = alert.showAndWait().orElse(somethingElse);

Alert resultDialog = new Alert(Alert.AlertType.INFORMATION,
                               answer.equals(result) ? "Correct" : "wrong",
                               ButtonType.OK);

resultDialog.show();
```

Lire Dialogues en ligne: https://riptutorial.com/fr/javafx/topic/3681/dialogues

Chapitre 8: Filetage

Examples

Mise à jour de l'interface utilisateur à l'aide de Platform.runLater

Les opérations de longue durée ne doivent pas être exécutées sur le thread d'application JavaFX, car cela empêche JavaFX de mettre à jour l'interface utilisateur, entraînant une interface utilisateur gelée.

En outre, toute modification d'un `Node` faisant partie d'un graphe de scène "en direct" **doit** se produire sur le thread d'application JavaFX. `Platform.runLater` peut être utilisé pour exécuter ces mises à jour sur le thread d'application JavaFX.

L'exemple suivant montre comment mettre à jour un `Node Text` plusieurs reprises à partir d'un thread différent:

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CounterApp extends Application {

    private int count = 0;
    private final Text text = new Text(Integer.toString(count));

    private void incrementCount() {
        count++;
        text.setText(Integer.toString(count));
    }

    @Override
    public void start(Stage primaryStage) {
        StackPane root = new StackPane();
        root.getChildren().add(text);

        Scene scene = new Scene(root, 200, 200);

        // longrunning operation runs on different thread
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {
                Runnable updater = new Runnable() {

                    @Override
                    public void run() {
                        incrementCount();
                    }
                };
```

```
                while (true) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ex) {
                    }

                    // UI update is run on the Application thread
                    Platform.runLater(updater);
                }
            }

        });
        // don't let thread prevent JVM shutdown
        thread.setDaemon(true);
        thread.start();

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Regroupement des mises à jour de l'interface utilisateur

Le code suivant ne répond plus pendant un court instant après le clic du bouton, car trop d'appels `Platform.runLater` sont utilisés. (Essayez de faire défiler le `ListView` immédiatement après le clic du bouton.)

```
@Override
public void start(Stage primaryStage) {
    ObservableList<Integer> data = FXCollections.observableArrayList();
    ListView<Integer> listView = new ListView<>(data);

    Button btn = new Button("Say 'Hello World'");
    btn.setOnAction((ActionEvent event) -> {
        new Thread(() -> {
            for (int i = 0; i < 100000; i++) {
                final int index = i;
                Platform.runLater(() -> data.add(index));
            }
        }).start();
    });

    Scene scene = new Scene(new VBox(listView, btn));

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Pour éviter cela au lieu d'utiliser un grand nombre de mises à jour, le code suivant utilise `AnimationTimer` pour exécuter la mise à jour une seule fois par image:

```
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.AnimationTimer;

public class Updater {

    @FunctionalInterface
    public static interface UpdateTask {

        public void update() throws Exception;
    }

    private final List<UpdateTask> updates = new ArrayList<>();

    private final AnimationTimer timer = new AnimationTimer() {

        @Override
        public void handle(long now) {
            synchronized (updates) {
                for (UpdateTask r : updates) {
                    try {
                        r.update();
                    } catch (Exception ex) {
                        Logger.getLogger(Updater.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                updates.clear();
                stop();
            }
        }
    };

    public void addTask(UpdateTask... tasks) {
        synchronized (updates) {
            updates.addAll(Arrays.asList(tasks));
            timer.start();
        }
    }

}
```

qui permet de regrouper les mises à jour à l'aide de la classe `Updater` :

```
private final Updater updater = new Updater();

...

        // Platform.runLater(() -> data.add(index));
        updater.addTask(() -> data.add(index));
```

Comment utiliser le service JavaFX

Au lieu d'exécuter des tâches intensives dans `JavaFX Thread` cela doit être fait dans un `Service` . Alors, qu'est-ce qu'un service ?

Un service est une classe qui crée un nouveau `Thread` chaque fois que vous le lancez et lui

transmet une tâche pour effectuer un travail. Le service peut renvoyer ou non une valeur.

Vous trouverez ci-dessous un exemple type de service JavaFX qui effectue un travail et retourne une `Map<String,String>()`:

```
public class WorkerService extends Service<Map<String, String>> {

    /**
     * Constructor
     */
    public WorkerService () {

        // if succeeded
        setOnSucceeded(s -> {
            //code if Service succeeds
        });

        // if failed
        setOnFailed(fail -> {
            //code it Service fails
        });

        //if cancelled
        setOnCancelled(cancelled->{
            //code if Service get's cancelled
        });
    }

    /**
    * This method starts the Service
    */
    public void startTheService(){
        if(!isRunning()){
           //...
           reset();
           start();
        }

    }

    @Override
    protected Task<Map<String, String>> createTask() {
        return new Task<Map<String, String>>() {
            @Override
            protected Void call() throws Exception {

                    //create a Map<String, String>
                    Map<String,String> map  = new HashMap<>();

                   //create other variables here

                   try{
                        //some code here
                        //.....do your manipulation here

                        updateProgress(++currentProgress, totalProgress);
                    }

                } catch (Exception ex) {
                    return null; //something bad happened so you have to do something instead
```

```
of returning null
                }

                return map;
            }
        };
    }

}
```

Lire Filetage en ligne: https://riptutorial.com/fr/javafx/topic/2230/filetage

Chapitre 9: FXML et contrôleurs

Syntaxe

- xmlns: fx = " http://javafx.com/fxml " // déclaration d'espace de noms

Examples

Exemple FXML

Un document FXML simple décrivant un `AnchorPane` contenant un bouton et un noeud d'étiquette:

```
<?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="com.example.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>
```

Cet exemple de fichier FXML est associé à une classe de contrôleur. L'association entre la classe FXML et la classe de contrôleur, dans ce cas, s'effectue en spécifiant le nom de la classe comme valeur de l'attribut `fx:controller` dans l'élément racine du fichier FXML: `fx:controller="com.example.FXMLDocumentController"` . La classe du contrôleur permet l'exécution du code Java en réponse aux actions de l'utilisateur sur les éléments d'interface utilisateur définis dans le fichier FXML:

```
package com.example ;

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;

public class FXMLDocumentController {

    @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 resources) {
        // Initialization code can go here.
        // The parameters url and resources can be omitted if they are not needed
    }

}
```

Un `FXMLLoader` peut être utilisé pour charger le fichier FXML:

```
public class MyApp extends Application {

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

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("FXMLDocument.fxml"));
        Parent root = loader.load();

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

}
```

La méthode de `load` effectue plusieurs actions et il est utile de comprendre l'ordre dans lequel elles se produisent. Dans cet exemple simple:

1. Le `FXMLLoader` lit et analyse le fichier FXML. Il crée des objets correspondant aux éléments définis dans le fichier et note tous les attributs `fx:id` définis sur ceux-ci.

2. L'élément racine du fichier FXML définissant un attribut `fx:controller` , `FXMLLoader` crée une *nouvelle instance* de la classe spécifiée. Par défaut, cela se produit en invoquant le constructeur sans argument sur la classe spécifiée.

3. Tous les éléments avec des attributs `fx:id` définis qui ont des champs dans le contrôleur avec des noms de champs correspondants, et qui sont `public` (non recommandés) ou annotés `@FXML` (recommandé) sont "injectés" dans ces champs correspondants. Donc, dans cet exemple, comme il y a une `Label` dans le fichier FXML avec `fx:id="label"` et un champ dans le contrôleur défini comme

   ```
   @FXML
   private Label label ;
   ```

   le champ `label` est initialisé avec l'instance `Label` créée par `FXMLLoader` .

4. Les gestionnaires d'événements sont enregistrés avec tous les éléments du fichier `onXXX="#..."` avec les `onXXX="#..."` définies. Ces gestionnaires d'événements appellent la méthode spécifiée dans la classe du contrôleur. Dans cet exemple, puisque le `Button` a `onAction="#handleButtonAction"` , et que le contrôleur définit une méthode

   ```
   @FXML
   private void handleButtonAction(ActionEvent event) { ... }
   ```

   Lorsqu'une action est déclenchée sur le bouton (par exemple, l'utilisateur appuie dessus), cette méthode est appelée. La méthode doit avoir un type de retour `void` et peut soit définir un paramètre correspondant au type d'événement (`ActionEvent` dans cet exemple), soit ne définir aucun paramètre.

5. Enfin, si la classe du contrôleur définit une méthode d' `initialize` , cette méthode est appelée. Notez que cela se produit une fois que les champs `@FXML` ont été injectés, ils peuvent donc être `@FXML` toute sécurité dans cette méthode et seront initialisés avec les instances correspondant aux éléments du fichier FXML. La méthode `initialize()` ne peut prendre aucun paramètre ou peut prendre une `URL` et un `ResourceBundle` . Dans ce dernier cas, ces paramètres seront renseignés par l' `URL` représentant l'emplacement du fichier FXML et tout ensemble `ResourceBundle` défini sur `FXMLLoader` via `loader.setResources(...)` . L'une ou l'autre peut être `null` si elles n'ont pas été définies.

Contrôleurs imbriqués

Il n'est pas nécessaire de créer l'interface utilisateur complète dans un seul fichier FXML à l'aide d'un seul contrôleur.

La `<fx:include>` peut être utilisée pour inclure un fichier fxml dans un autre. Le contrôleur du fichier fxml inclus peut être injecté dans le contrôleur du fichier inclus, comme tout autre objet créé par `FXMLLoader` .

Cela se fait en ajoutant l'attribut `fx:id` à l'élément `<fx:include>` . De cette façon, le contrôleur du fichier fxml inclus sera injecté dans le champ avec le nom `<fx:id value>Controller` .

Exemples:

fx: valeur de l'identifiant	nom du champ pour l'injection
foo	fooController
réponse42	answer42Controller
xYz	xYzController

Échantillon de fxml

Compteur

Ceci est un fichier fxml contenant un `StackPane` avec un noeud `Text` . Le contrôleur de ce fichier fxml permet d'obtenir la valeur actuelle du compteur et d'incrémenter le compteur:

counter.fxml

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

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<StackPane prefHeight="200" prefWidth="200" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="counter.CounterController">
    <children>
        <Text fx:id="counter" />
    </children>
</StackPane>
```

CounterController

```
package counter;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class CounterController {
    @FXML
    private Text counter;

    private int value = 0;

    public void initialize() {
        counter.setText(Integer.toString(value));
    }

    public void increment() {
        value++;
        counter.setText(Integer.toString(value));
    }

    public int getValue() {
        return value;
    }

}
```

Y compris fxml

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

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane prefHeight="500" prefWidth="500" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="counter.OuterController">
    <left>
        <Button BorderPane.alignment="CENTER" text="increment" onAction="#increment" />
```

```
    </left>
    <center>
        <!-- content from counter.fxml included here -->
        <fx:include fx:id="count" source="counter.fxml" />
    </center>
</BorderPane>
```

Contrôleur externe

Le contrôleur du fichier fxml inclus est injecté dans ce contrôleur. Ici, le gestionnaire de l'événement `onAction` pour le `Button` est utilisé pour incrémenter le compteur.

```
package counter;

import javafx.fxml.FXML;

public class OuterController {

    // controller of counter.fxml injected here
    @FXML
    private CounterController countController;

    public void initialize() {
        // controller available in initialize method
        System.out.println("Current value: " + countController.getValue());
    }

    @FXML
    private void increment() {
        countController.increment();
    }

}
```

Les fxml peuvent être chargés comme ceci, en supposant que le code est appelé depuis une classe dans le même paquet que `outer.fxml` :

```
Parent parent = FXMLLoader.load(getClass().getResource("outer.fxml"));
```

Définir des blocs et

Parfois, un élément doit être créé en dehors de la structure d'objet habituelle dans le fichier fxml.

C'est ici *qu'interviennent les blocs de définition* :

Les contenus contenus dans un élément `<fx:define>` ne sont pas ajoutés à l'objet créé pour l'élément parent.

Chaque élément enfant du `<fx:define>` besoin d'un attribut `fx:id` .

Les objets créés de cette manière peuvent être référencés ultérieurement en utilisant l'élément `<fx:reference>` ou en utilisant la liaison d'expression.

L'élément `<fx:reference>` peut être utilisé pour référencer un élément avec un attribut `fx:id` qui est

géré avant que l'élément `<fx:reference>` soit géré en utilisant la même valeur que l'attribut `fx:id` de l'élément référencé dans le attribut `source` de l'élément `<fx:reference>` .

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

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


<VBox xmlns:fx="http://javafx.com/fxml/1" prefHeight="300.0" prefWidth="300.0"
xmlns="http://javafx.com/javafx/8">
    <children>
        <fx:define>
            <String fx:value="My radio group" fx:id="text" />
        </fx:define>
        <Text>
            <text>
                <!-- reference text defined above using fx:reference -->
                <fx:reference source="text"/>
            </text>
        </Text>
        <RadioButton text="Radio 1">
            <toggleGroup>
                <ToggleGroup fx:id="group" />
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 2">
            <toggleGroup>
                <!-- reference ToggleGroup created for last RadioButton -->
                <fx:reference source="group"/>
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 3" toggleGroup="$group" />

        <!-- reference text defined above using expression binding -->
        <Text text="$text" />
    </children>
</VBox>
```

Passer des données à FXML - accéder au contrôleur existant

Problème: Certaines données doivent être transmises à une scène chargée à partir d'un fichier fxml.

Solution

Spécifiez un contrôleur à l'aide de l'attribut `fx:controller` et obtenez l'instance de contrôleur créée lors du chargement à partir de l'instance `FXMLLoader` utilisée pour charger le fichier fxml.

Ajoutez des méthodes pour transmettre les données à l'instance de contrôleur et gérez les données dans ces méthodes.

FXML

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

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Manette

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    @FXML
    private Text target;

    public void setData(String data) {
        target.setText(data);
    }

}
```

Code utilisé pour charger le fichier fxml

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));
Parent root = loader.load();
TestController controller = loader.<TestController>getController();
controller.setData(data);
```

Transmission de données à FXML - Spécification de l'instance de contrôleur

Problème: Certaines données doivent être transmises à une scène chargée à partir d'un fichier fxml.

Solution

Définissez le contrôleur à l'aide de l'instance `FXMLLoader` utilisée ultérieurement pour charger le fichier fxml.

Assurez-vous que le contrôleur contient les données pertinentes avant de charger le fichier fxml.

Remarque: dans ce cas, le fichier fxml ne doit pas contenir l'attribut `fx:controller` .

FXML

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

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Manette

```
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Code utilisé pour charger le fichier fxml

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

TestController controller = new TestController(data);
loader.setController(controller);

Parent root = loader.load();
```

Passer des paramètres à FXML - en utilisant un controllerFactory

Problème: Certaines données doivent être transmises à une scène chargée à partir d'un fichier fxml.

Solution

Spécifiez une fabrique de contrôleurs responsable de la création des contrôleurs. Transmettez les données à l'instance de contrôleur créée par la fabrique.

FXML

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

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Manette

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Code utilisé pour charger le fichier fxml

String data = "Hello World!"

```
Map<Class, Callable<?>> creators = new HashMap<>();
creators.put(TestController.class, new Callable<TestController>() {

    @Override
    public TestController call() throws Exception {
        return new TestController(data);
    }

});

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

loader.setControllerFactory(new Callback<Class<?>, Object>() {

    @Override
    public Object call(Class<?> param) {
```

```
        Callable<?> callable = creators.get(param);
        if (callable == null) {
            try {
                // default handling: use no-arg constructor
                return param.newInstance();
            } catch (InstantiationException | IllegalAccessException ex) {
                throw new IllegalStateException(ex);
            }
        } else {
            try {
                return callable.call();
            } catch (Exception ex) {
                throw new IllegalStateException(ex);
            }
        }
    }
});

Parent root = loader.load();
```

Cela peut sembler complexe, mais cela peut être utile si le fxml doit pouvoir décider quelle classe de contrôleur il a besoin.

Création d'instance dans FXML

La classe suivante est utilisée pour démontrer comment créer des instances de classes:

JavaFX 8

L'annotation dans `Person(@NamedArg("name") String name)` doit être supprimée car l'annotation `@NamedArg` est indisponible.

```
package fxml.sample;

import javafx.beans.NamedArg;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public static final Person JOHN = new Person("John");

    public Person() {
        System.out.println("Person()");
    }

    public Person(@NamedArg("name") String name) {
        System.out.println("Person(String)");
        this.name.set(name);
    }

    public Person(Person person) {
        System.out.println("Person(Person)");
        this.name.set(person.getName());
    }

    private final StringProperty name = new SimpleStringProperty();
```

```
    public final String getName() {
        System.out.println("getter");
        return this.name.get();
    }

    public final void setName(String value) {
        System.out.println("setter");
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        System.out.println("property getter");
        return this.name;
    }

    public static Person valueOf(String value) {
        System.out.println("valueOf");
        return new Person(value);
    }

    public static Person createPerson() {
        System.out.println("createPerson");
        return new Person();
    }

}
```

Supposons que la classe `Person` a déjà été initialisée avant de charger le fichier fxml.

Une note sur les importations

Dans l'exemple fxml suivant, la section des importations sera omise. Cependant, le fxml devrait commencer par

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

suivie par une section d'importation important toutes les classes utilisées dans le fichier fxml. Ces importations sont similaires aux importations non statiques, mais sont ajoutées comme instructions de traitement. **Même les classes du package `java.lang` doivent être importées.**

Dans ce cas, les importations suivantes doivent être ajoutées:

```
<?import java.lang.*?>
<?import fxml.sample.Person?>
```

JavaFX 8

`@NamedArg` constructeur annoté

S'il existe un constructeur où chaque paramètre est annoté avec `@NamedArg` et que toutes les valeurs des annotations `@NamedArg` sont présentes dans le fichier fxml, le constructeur sera utilisé avec ces paramètres.

```
<Person name="John"/>
```

```
<Person xmlns:fx="http://javafx.com/fxml">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Les deux résultent dans la sortie de console suivante, si chargée:

```
Person(String)
```

Aucun constructeur args

S'il n'y a pas de constructeur annoté `@NamedArg` approprié disponible, le constructeur qui ne prend aucun paramètre sera utilisé.

Supprimez l'annotation `@NamedArg` du constructeur et essayez de charger.

```
<Person name="John"/>
```

Cela utilisera le constructeur sans paramètres.

Sortie:

```
Person()
setter
```

attribut `fx:value`

L'attribut `fx:value` peut être utilisé pour transmettre sa valeur à une méthode `valueOf static` prenant un paramètre `String` et renvoyant l'instance à utiliser.

Exemple

```
<Person xmlns:fx="http://javafx.com/fxml" fx:value="John"/>
```

Sortie:

```
valueOf
Person(String)
```

`fx:factory`

L'attribut `fx:factory` permet de créer des objets en utilisant des méthodes `static` arbitraires qui ne prennent pas de paramètres.

Exemple

```
<Person xmlns:fx="http://javafx.com/fxml" fx:factory="createPerson">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Sortie:

```
createPerson
Person()
setter
```

<fx:copy>

En utilisant `fx:copy` un constructeur de copie peut être appelé. Spécifier le `fx:id` d'un autre L'attribut `source` de la balise invoquera le constructeur de copie avec cet objet comme paramètre.

Exemple:

```
<ArrayList xmlns:fx="http://javafx.com/fxml">
    <Person fx:id="p1" fx:constant="JOHN"/>
    <fx:copy source="p1"/>
</ArrayList>
```

Sortie

```
Person(Person)
getter
```

fx:constant

`fx:constant` permet d'obtenir une valeur à partir d'un `static final` .

Exemple

```
<Person xmlns:fx="http://javafx.com/fxml" fx:constant="JOHN"/>
```

ne produira aucune sortie, car cela fait juste référence à `JOHN` qui a été créé lors de l'initialisation de la classe.

Définition des propriétés

Il existe plusieurs manières d'ajouter des données à un objet dans fxml:

<property>

Une balise avec le nom d'une propriété peut être ajoutée en tant qu'enfant d'un élément utilisé pour créer une instance. L'enfant de cette balise est affecté à la propriété à l'aide du setter ou

ajouté au contenu de la propriété (liste / propriétés en lecture seule).

Propriété par défaut

Une classe peut être annotée avec l'annotation `@DefaultProperty` . Dans ce cas, les éléments peuvent être ajoutés directement en tant qu'élément enfant sans utiliser d'élément portant le nom de la propriété.

`property="value"` attribut `property="value"`

Les propriétés peuvent être affectées en utilisant le nom de la propriété comme nom d'attribut et la valeur comme valeur d'attribut. Cela a le même effet que d'ajouter l'élément suivant en tant qu'enfant de la balise:

```
<property>
    <String fx:value="value" />
</property>
```

dispositifs statiques

Les propriétés peuvent également être définies à l'aide de paramètres `static` . Ce sont `static` méthodes `static` nommées `setProperty` qui prennent l'élément comme premier paramètre et la valeur à définir comme second paramètre. Ces méthodes peuvent être utilisées dans n'importe quelle classe et peuvent être utilisées avec `ContainingClass.property` au lieu du nom de propriété habituel.

Note: Actuellement, il semble nécessaire d'avoir une méthode getter statique correspondante (c'est-à-dire une méthode statique appelée `getProperty` prenant l'élément comme paramètre dans la même classe que le setter statique) pour que cela fonctionne à moins que le type de valeur soit `String` .

Type de coercition

Le mécanisme suivant est utilisé pour obtenir un objet de la classe correcte lors des affectations, par exemple pour s'adapter au type de paramètre d'une méthode de réglage.

Si les classes sont assignables, la valeur elle-même est utilisée.

Sinon, la valeur est convertie comme suit

Type de cible	valeur utilisée (valeur source `s`)
`Boolean` `boolean`	`Boolean.valueOf(s)`

Type de cible	valeur utilisée (valeur source `s`)
`char` , `Character`	`s.toString.charAt(0)`
autre type primitif ou type de wrapper	méthode appropriée pour le type de cible, dans le cas où le `s` est un `Number` , le `valueOf(s.toString())` pour le type de wrapper sinon
`BigInteger`	`BigInteger.valueOf(s.longValue())` is `s` est un `Number` , `new BigInteger(s.toString())` sinon
`BigDecimal`	`BigDecimal.valueOf(s.doubleValue())` is `s` est un `Number` , `new BigDecimal(s.toString())` sinon
Nombre	`Double.valueOf(s.toString())` si `s.toString()` contient un `.` , `Long.valueOf(s.toString())` sinon
`Class`	`Class.forName(s.toString())` à l'aide du contexte `ClassLoader` du thread en cours sans initialiser la classe
enum	Le résultat de la méthode `valueOf` , converti en une `String` tout en majuscule séparée par _ inséré avant chaque lettre majuscule, si `s` est une `String` qui commence par une lettre minuscule
autre	la valeur renvoyée par une méthode `valueOf static` dans le targetType, qui a un paramètre correspondant au type de `s` ou une super-classe de ce type

Remarque: Ce comportement n'est pas bien documenté et pourrait être sujet à modification.

Exemple

```
public enum Location {
    WASHINGTON_DC,
    LONDON;
}
```

```
package fxml.sample;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javafx.beans.DefaultProperty;

@DefaultProperty("items")
public class Sample {

    private Location loaction;
```

```
    public Location getLoaction() {
        return loaction;
    }

    public void setLoaction(Location loaction) {
        this.loaction = loaction;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    int number;

    private final List<Object> items = new ArrayList<>();

    public List<Object> getItems() {
        return items;
    }

    private final Map<String, Object> map = new HashMap<>();

    public Map<String, Object> getMap() {
        return map;
    }

    private BigInteger serialNumber;

    public BigInteger getSerialNumber() {
        return serialNumber;
    }

    public void setSerialNumber(BigInteger serialNumber) {
        this.serialNumber = serialNumber;
    }

    @Override
    public String toString() {
        return "Sample{" + "loaction=" + loaction + ", number=" + number + ", items=" + items
+ ", map=" + map + ", serialNumber=" + serialNumber + '}';
    }

}
```

```
package fxml.sample;

public class Container {

    public static int getNumber(Sample sample) {
        return sample.number;
    }

    public static void setNumber(Sample sample, int number) {
        sample.number = number;
    }

    private final String value;
```

```
    private Container(String value) {
        this.value = value;
    }

    public static Container valueOf(String s) {
        return new Container(s);
    }

    @Override
    public String toString() {
        return "42" + value;
    }

}
```

Impression du résultat du chargement du fichier `fxml` ci-dessous

```
Sample{loaction=WASHINGTON_DC, number=5, items=[42a, 42b, 42c, 42d, 42e, 42f], map={answer=42,
g=9.81, hello=42A, sample=Sample{loaction=null, number=33, items=[], map={},
serialNumber=null}}, serialNumber=4299}
```

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

<?import java.lang.*?>
<?import fxml.sample.*?>

<Sample xmlns:fx="http://javafx.com/fxml/1" Container.number="5" loaction="washingtonDc">

    <!-- set serialNumber property (type coercion) -->
    <serialNumber>
        <Container fx:value="99"/>
    </serialNumber>

    <!-- Add elements to default property-->
    <Container fx:value="a"/>
    <Container fx:value="b"/>
    <Container fx:value="c"/>
    <Container fx:value="d"/>
    <Container fx:value="e"/>
    <Container fx:value="f"/>

    <!-- fill readonly map property -->
    <map g="9.81">
        <hello>
            <Container fx:value="A"/>
        </hello>
        <answer>
            <Container fx:value=""/>
        </answer>
        <sample>
            <Sample>
                <!-- static setter-->
                <Container.number>
                    <Integer fx:value="33" />
                </Container.number>
            </Sample>
        </sample>
    </map>
```

```
</Sample>
```

Lire FXML et contrôleurs en ligne: https://riptutorial.com/fr/javafx/topic/1580/fxml-et-controleurs

Chapitre 10: Graphique

Examples

Camembert

La classe `PieChart` dessine des données sous la forme d'un cercle divisé en tranches. Chaque tranche représente un pourcentage (partie) pour une valeur particulière. Les données du graphique en `PieChart.Data` sont `PieChart.Data` dans des objets `PieChart.Data` . Chaque objet `PieChart.Data` a deux champs: le nom de la tranche et sa valeur correspondante.

Constructeurs

Pour créer un graphique en secteurs, nous devons créer l'objet de la classe `PieChart` . Deux constructeurs sont à notre disposition. L'un d'eux crée un graphique vide qui n'affichera rien à moins que les données ne soient définies avec la méthode `setData` :

```
PieChart pieChart = new PieChart(); // Creates an empty pie chart
```

Et le second nécessite une `ObservableList` de `PieChart.Data` à transmettre en paramètre.

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Cats", 50),
            new PieChart.Data("Dogs", 50));
PieChart pieChart(valueList); // Creates a chart with the given data
```

Les données

Les valeurs des tranches ne doivent pas nécessairement totaliser 100, car la taille de la tranche sera calculée proportionnellement à la somme de toutes les valeurs.

L'ordre dans lequel les entrées de données sont ajoutées à la liste déterminera leur position sur le graphique. Par défaut, ils sont disposés dans le sens des aiguilles d'une montre, mais ce comportement peut être inversé:

```
pieChart.setClockwise(false);
```

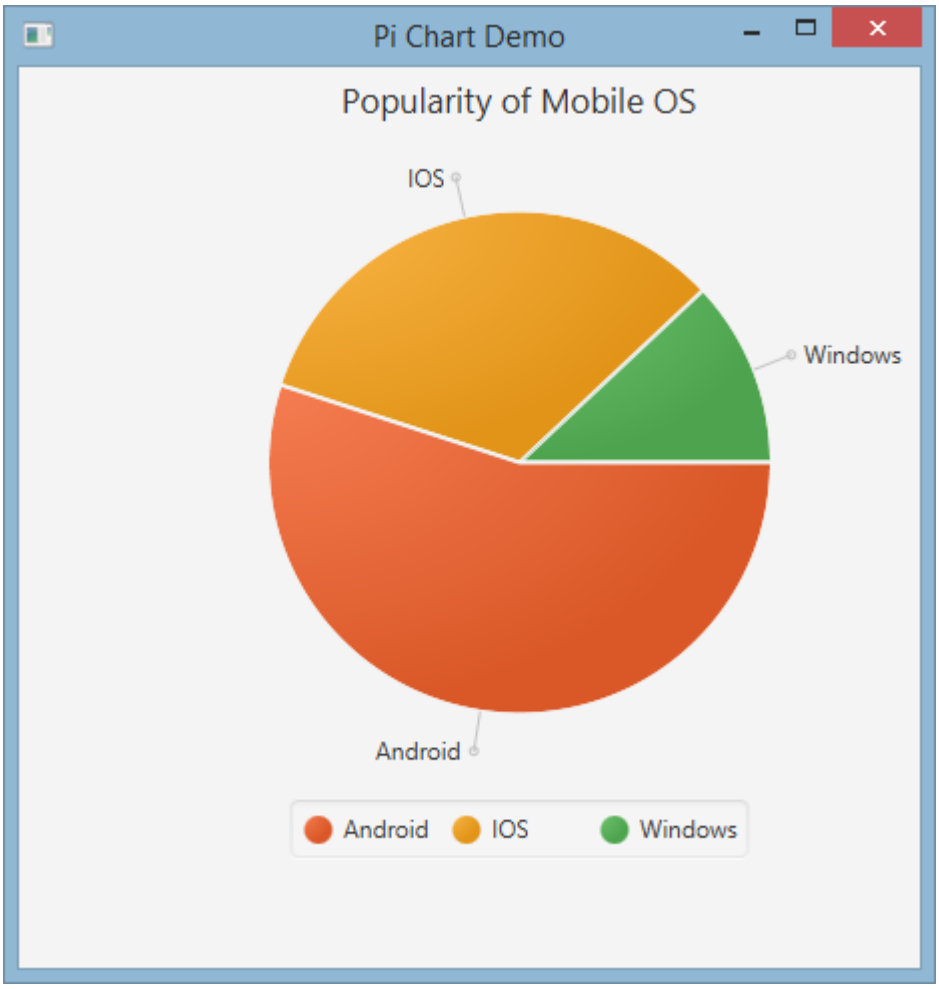
Exemple

L'exemple suivant crée un graphique circulaire simple:

```
import javafx.application.Application;
```

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class Main extends Application {

@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();
    ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Android", 55),
            new PieChart.Data("IOS", 33),
            new PieChart.Data("Windows", 12));
    // create a pieChart with valueList data.
    PieChart pieChart = new PieChart(valueList);
    pieChart.setTitle("Popularity of Mobile OS");
    //adding pieChart to the root.
    root.getChildren().addAll(pieChart);
    Scene scene = new Scene(root, 450, 450);

    primaryStage.setTitle("Pie Chart Demo");
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

Sortie:



Graphique à secteurs interactif

Par défaut, `PieChart` ne gère aucun événement, mais ce comportement peut être modifié car chaque segment est un `Node` JavaFX.

Dans l'exemple ci-dessous, nous initialisons les données, nous les affectons au graphique, puis nous parcourons l'ensemble de données en ajoutant des info-bulles à chaque tranche, afin que les valeurs, normalement masquées, puissent être présentées à l'utilisateur.

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Nitrogen", 7809),
            new PieChart.Data("Oxygen", 2195),
            new PieChart.Data("Other", 93));

PieChart pieChart = new PieChart(valueList);
pieChart.setTitle("Air composition");

pieChart.getData().forEach(data -> {
    String percentage = String.format("%.2f%%", (data.getPieValue() / 100));
    Tooltip toolTip = new Tooltip(percentage);
    Tooltip.install(data.getNode(), toolTip);
});
```

Graphique en ligne

La classe `LineChart` présente les données sous la forme d'une série de points de données connectés par des lignes droites. Chaque point de données est `XYChart.Data` dans `XYChart.Data` objet `XYChart.Data` et les points de données sont regroupés dans `XYChart.Series` .

Chaque objet `XYChart.Data` possède deux champs, accessibles via `getXValue` et `getYValue` , correspondant à une valeur x et ay sur un graphique.

```
XYChart.Data data = new XYChart.Data(1,3);
System.out.println(data.getXValue()); // Will print 1
System.out.println(data.getYValue()); // Will print 3
```

Les axes

Avant de créer un `LineChart` nous devons définir ses axes. Par exemple, le constructeur par défaut sans argument d'une classe `NumberAxis` créera un axe de mesure automatique prêt à l'emploi et ne nécessitant aucune configuration supplémentaire.

```
Axis xAxis = new NumberAxis();
```

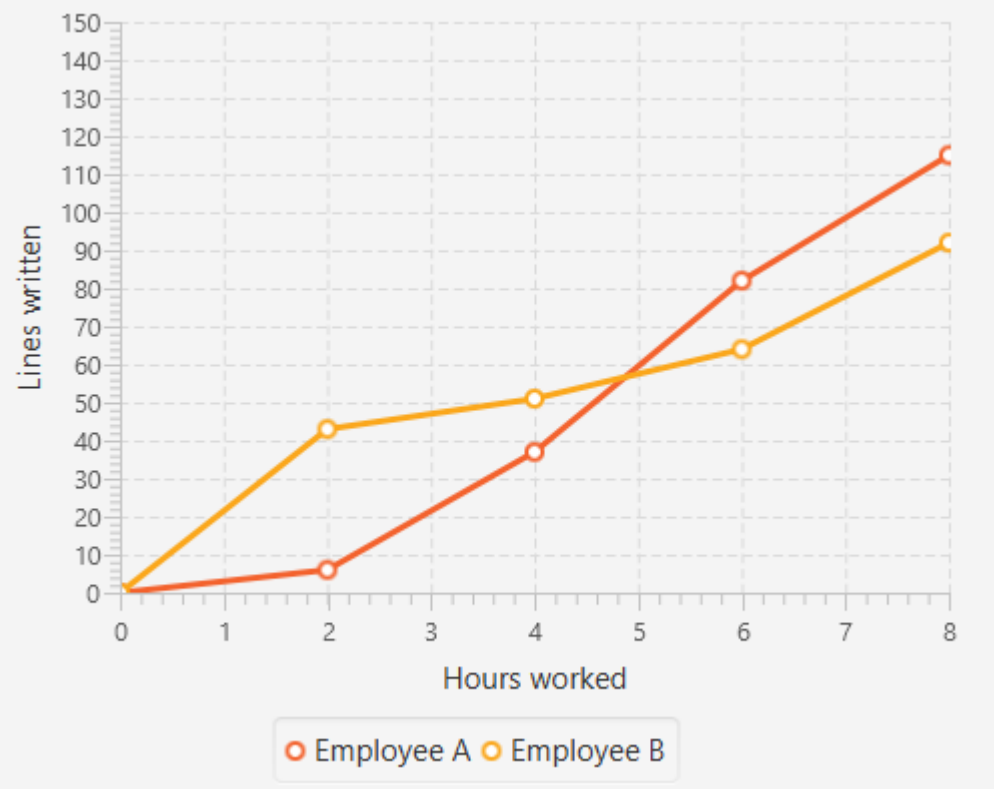
Exemple

Dans l'exemple complet ci-dessous, nous créons deux séries de données qui seront affichées sur le même graphique. Les étiquettes, les plages et les valeurs de tick des axes sont explicitement définies.

```
@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();

    // Create empty series
    ObservableList<XYChart.Series> seriesList = FXCollections.observableArrayList();

    // Create data set for the first employee and add it to the series
    ObservableList<XYChart.Data> aList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 6),
            new XYChart.Data(4, 37),
            new XYChart.Data(6, 82),
            new XYChart.Data(8, 115)
    );
    seriesList.add(new XYChart.Series("Employee A", aList));

    // Create data set for the second employee and add it to the series
    ObservableList<XYChart.Data> bList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 43),
            new XYChart.Data(4, 51),
            new XYChart.Data(6, 64),
            new XYChart.Data(8, 92)
    );
```

```
    seriesList.add(new XYChart.Series("Employee B", bList));

    // Create axes
    Axis xAxis = new NumberAxis("Hours worked", 0, 8, 1);
    Axis yAxis = new NumberAxis("Lines written", 0, 150, 10);

    LineChart chart = new LineChart(xAxis, yAxis, seriesList);

    root.getChildren().add(chart);

    Scene scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

Sortie:



Lire Graphique en ligne: https://riptutorial.com/fr/javafx/topic/2631/graphique

Chapitre 11: Impression

Examples

Impression de base

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.printPage(some-node);
    if (success) {
        pJ.endJob();
    }
}
```

Cela imprime à l'imprimante par défaut sans afficher aucune boîte de dialogue à l'utilisateur. Pour utiliser une imprimante autre que celle par défaut, vous pouvez utiliser `PrinterJob#createPrinterJob(Printer)` pour définir l'imprimante actuelle. Vous pouvez l'utiliser pour afficher toutes les imprimantes sur votre système:

```
System.out.println(Printer.getAllPrinters());
```

Impression avec la boîte de dialogue système

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.showPrintDialog(primaryStage);// this is the important line
    if (success) {
        pJ.endJob();
    }
}
```

Lire Impression en ligne: https://riptutorial.com/fr/javafx/topic/5157/impression

Chapitre 12: Internationalisation en JavaFX

Examples

Chargement d'un ensemble de ressources

JavaFX fournit un moyen facile d'internationaliser vos interfaces utilisateur. Lors de la création d'une vue à partir d'un fichier FXML, vous pouvez fournir à `FXMLLoader` un regroupement de ressources:

```
Locale locale = new Locale("en", "UK");
ResourceBundle bundle = ResourceBundle.getBundle("strings", locale);

Parent root = FXMLLoader.load(getClass().getClassLoader()
                                .getResource("ui/main.fxml"), bundle);
```

Ce bundle fourni est automatiquement utilisé pour traduire tous les textes de votre fichier FXML qui commencent par un `%` . Disons que votre fichier de propriétés `strings_en_UK.properties` contient la ligne suivante:

```
ui.button.text=I'm a Button
```

Si vous avez une définition de bouton dans votre fichier FXML comme ceci:

```
<Button text="%ui.button.text"/>
```

Il recevra automatiquement la traduction de la clé `ui.button.text` .

Manette

Un regroupement de ressources contient des objets spécifiques aux paramètres régionaux. Vous pouvez transmettre le bundle à `FXMLLoader` lors de sa création. Le contrôleur doit implémenter l'interface `Initializable` et remplacer la méthode d' `initialize(URL location, ResourceBundle resources)` . Le second paramètre de cette méthode est `ResourceBundle` qui est transmis du FXMLLoader au contrôleur et peut être utilisé par le contrôleur pour traduire davantage de texte ou modifier d'autres informations dépendantes de la localisation.

```
public class MyController implements Initializable {

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        label.setText(resources.getString("country"));
    }
}
```

Changer de langue de manière dynamique lorsque l'application est en cours

d'exécution

Cet exemple montre comment créer une application JavaFX, dans laquelle le langage peut être changé de manière dynamique pendant l'exécution de l'application.

Voici les fichiers de regroupement de messages utilisés dans l'exemple:

messages_en.properties :

```
window.title=Dynamic language change
button.english=English
button.german=German
label.numSwitches=Number of language switches: {0}
```

messages_de.properties :

```
window.title=Dynamischer Sprachwechsel
button.english=Englisch
button.german=Deutsch
label.numSwitches=Anzahl Sprachwechsel: {0}
```

L'idée de base est d'avoir une classe d'utilitaire I18N (en alternative, cela pourrait être un singleton).

```
import javafx.beans.binding.Bindings;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.concurrent.Callable;

/**
 * I18N utility class..
 */
public final class I18N {

    /** the current selected Locale. */
    private static final ObjectProperty<Locale> locale;

    static {
        locale = new SimpleObjectProperty<>(getDefaultLocale());
        locale.addListener((observable, oldValue, newValue) -> Locale.setDefault(newValue));
    }

    /**
     * get the supported Locales.
     *
     * @return List of Locale objects.
```

```
     */
    public static List<Locale> getSupportedLocales() {
        return new ArrayList<>(Arrays.asList(Locale.ENGLISH, Locale.GERMAN));
    }

    /**
     * get the default locale. This is the systems default if contained in the supported
locales, english otherwise.
     *
     * @return
     */
    public static Locale getDefaultLocale() {
        Locale sysDefault = Locale.getDefault();
        return getSupportedLocales().contains(sysDefault) ? sysDefault : Locale.ENGLISH;
    }

    public static Locale getLocale() {
        return locale.get();
    }

    public static void setLocale(Locale locale) {
        localeProperty().set(locale);
        Locale.setDefault(locale);
    }

    public static ObjectProperty<Locale> localeProperty() {
        return locale;
    }

    /**
     * gets the string with the given key from the resource bundle for the current locale and
uses it as first argument
     * to MessageFormat.format, passing in the optional args and returning the result.
     *
     * @param key
     *         message key
     * @param args
     *         optional arguments for the message
     * @return localized formatted string
     */
    public static String get(final String key, final Object... args) {
        ResourceBundle bundle = ResourceBundle.getBundle("messages", getLocale());
        return MessageFormat.format(bundle.getString(key), args);
    }

    /**
     * creates a String binding to a localized String for the given message bundle key
     *
     * @param key
     *         key
     * @return String binding
     */
    public static StringBinding createStringBinding(final String key, Object... args) {
        return Bindings.createStringBinding(() -> get(key, args), locale);
    }

    /**
     * creates a String Binding to a localized String that is computed by calling the given
func
     *
     * @param func
```

```
     *         function called on every change
     * @return StringBinding
     */
    public static StringBinding createStringBinding(Callable<String> func) {
        return Bindings.createStringBinding(func, locale);
    }

    /**
     * creates a bound Label whose value is computed on language change.
     *
     * @param func
     *         the function to compute the value
     * @return Label
     */
    public static Label labelForValue(Callable<String> func) {
        Label label = new Label();
        label.textProperty().bind(createStringBinding(func));
        return label;
    }

    /**
     * creates a bound Button for the given resourcebundle key
     *
     * @param key
     *         ResourceBundle key
     * @param args
     *         optional arguments for the message
     * @return Button
     */
    public static Button buttonForKey(final String key, final Object... args) {
        Button button = new Button();
        button.textProperty().bind(createStringBinding(key, args));
        return button;
    }
}
```

Cette classe a un `locale` champ statique qui est un objet Java `Locale` enveloppé dans JavaFX `ObjectProperty` , afin que des liaisons puissent être créées pour cette propriété. Les premières méthodes sont les méthodes standard pour obtenir et définir une propriété JavaFX.

La méthode `get(final String key, final Object... args)` est la méthode de base utilisée pour l'extraction réelle d'un message à partir d'un `ResourceBundle` .

Les deux méthodes nommées `createStringBinding` créent un objet `StringBinding` lié au champ des `locale` . Par conséquent, les liaisons changent chaque fois que la propriété `locale` change. Le premier utilise ses arguments pour récupérer et formater un message en utilisant la méthode `get` mentionnée ci-dessus, le second est passé dans un `Callable` , qui doit produire la nouvelle valeur de chaîne.

Les deux dernières méthodes sont des méthodes pour créer des composants JavaFX. La première méthode est utilisée pour créer une `Label` et utilise un `Callable` pour sa liaison de chaîne interne. Le second crée un `Button` et utilise une valeur de clé pour la récupération de la liaison String.

Bien sûr, de nombreux objets différents pourraient être créés comme `MenuItem` ou `ToolTip` mais ces deux éléments devraient suffire pour donner un exemple.

Ce code montre comment cette classe est utilisée dans l'application:

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

import java.util.Locale;

/**
 * Sample application showing dynamic language switching,
 */
public class I18nApplication extends Application {

    /** number of language switches. */
    private Integer numSwitches = 0;

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.titleProperty().bind(I18N.createStringBinding("window.title"));

        // create content
        BorderPane content = new BorderPane();

        // at the top two buttons
        HBox hbox = new HBox();
        hbox.setPadding(new Insets(5, 5, 5, 5));
        hbox.setSpacing(5);

        Button buttonEnglish = I18N.buttonForKey("button.english");
        buttonEnglish.setOnAction((evt) -> switchLanguage(Locale.ENGLISH));
        hbox.getChildren().add(buttonEnglish);

        Button buttonGerman = I18N.buttonForKey("button.german");
        buttonGerman.setOnAction((evt) -> switchLanguage(Locale.GERMAN));
        hbox.getChildren().add(buttonGerman);

        content.setTop(hbox);

        // a label to display the number of changes, recalculating the text on every change
        final Label label = I18N.labelForValue(() -> I18N.get("label.numSwitches",
numSwitches));
        content.setBottom(label);

        primaryStage.setScene(new Scene(content, 400, 200));
        primaryStage.show();
    }

    /**
     * sets the given Locale in the I18N class and keeps count of the number of switches.
     *
     * @param locale
     *         the new local to set
     */
    private void switchLanguage(Locale locale) {
        numSwitches++;
```

```
            I18N.setLocale(locale);
        }
    }
```

L'application montre trois manières différentes d'utiliser le `StringBinding` créé par la classe `I18N` :

1. le titre de la fenêtre est lié en utilisant directement un `StringBinding` .
2. les boutons utilisent la méthode d'assistance avec les clés de message
3. l'étiquette utilise la méthode d'assistance avec un `Callable` . Ce `Callable` utilise la méthode `I18N.get()` pour obtenir une chaîne traduite formatée contenant le nombre réel de commutateurs.

En cliquant sur un bouton, le compteur est augmenté et la propriété locale de `I18N` est définie, ce qui déclenche la modification des liaisons de chaînes et définit ainsi la chaîne de l'interface utilisateur sur de nouvelles valeurs.

Lire Internationalisation en JavaFX en ligne:
https://riptutorial.com/fr/javafx/topic/5434/internationalisation-en-javafx

Chapitre 13: les fenêtres

Examples

Créer une nouvelle fenêtre

Pour afficher du contenu dans une nouvelle fenêtre, une `Stage` doit être créée. Après la création et l'initialisation, `show` ou `showAndWait` doit être appelé sur l'objet `Stage` :

```
// create sample content
Rectangle rect = new Rectangle(100, 100, 200, 300);
Pane root = new Pane(rect);
root.setPrefSize(500, 500);

Parent content = root;

// create scene containing the content
Scene scene = new Scene(content);

Stage window = new Stage();
window.setScene(scene);

// make window visible
window.show();
```

Remarque: Ce code doit être exécuté sur le thread d'application JavaFX.

Création d'une boîte de dialogue personnalisée

Vous pouvez créer des boîtes de dialogue personnalisées contenant de nombreux composants et y exécuter de nombreuses fonctionnalités. Il se comporte comme une deuxième étape sur la scène du propriétaire.
Dans l'exemple suivant, une application qui affiche une personne dans la tableview de la scène principale et crée une personne dans une boîte de dialogue (AddingPersonDialog) préparée. Les interfaces graphiques créées par SceneBuilder, mais elles peuvent être créées par des codes java purs.

Exemple d'application:

AppMain.java

```
package customdialog;

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

public class AppMain extends Application {
```

```
    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
    @FXML
    private TableColumn colName;
    @FXML
    private TableColumn colAge;

    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
    }
```

```
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
               </columnResizePolicy>
            </TableView>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {

    @FXML
```

```
    private TextField tfId;

    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

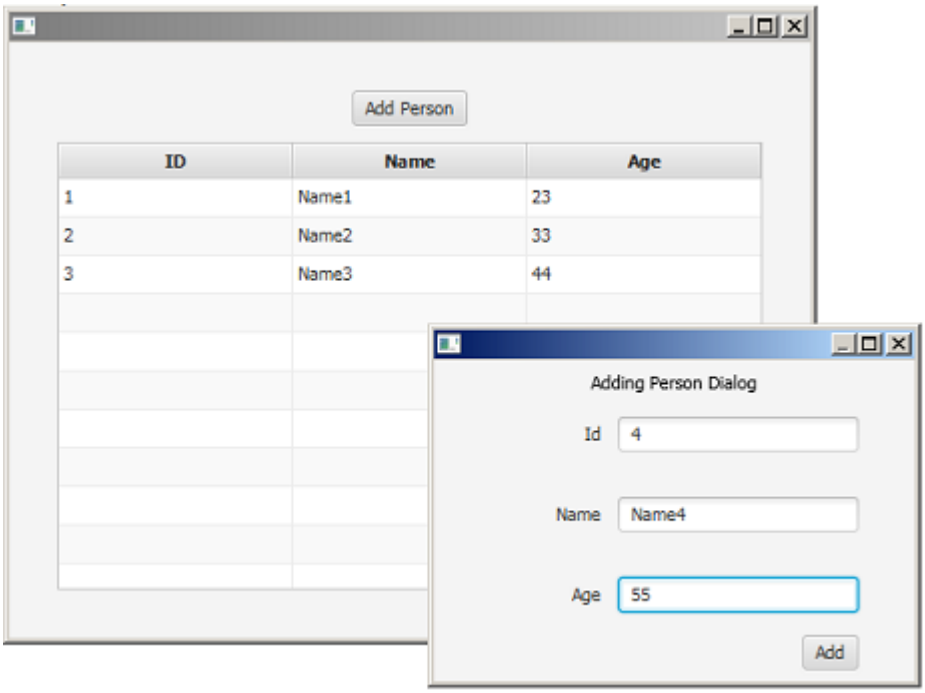
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
```

```
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }
```

```
    public int getId() {
        return id.get();
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```



Capture d'écran

Création d'une boîte de dialogue personnalisée

Vous pouvez créer des boîtes de dialogue personnalisées contenant de nombreux composants et y exécuter de nombreuses fonctionnalités. Il se comporte comme une deuxième étape sur la scène du propriétaire.

Dans l'exemple suivant, une application qui affiche une personne dans la tableview de la scène principale et crée une personne dans une boîte de dialogue (AddingPersonDialog) préparée. Les interfaces graphiques créées par SceneBuilder, mais elles peuvent être créées par des codes java purs.

Exemple d'application:

AppMain.java

```
package customdialog;

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

public class AppMain extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
    @FXML
```

```
    private TableColumn colName;
    @FXML
    private TableColumn colAge;

    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
               </columnResizePolicy>
```

```
            </TableView>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {

    @FXML
    private TextField tfId;

    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

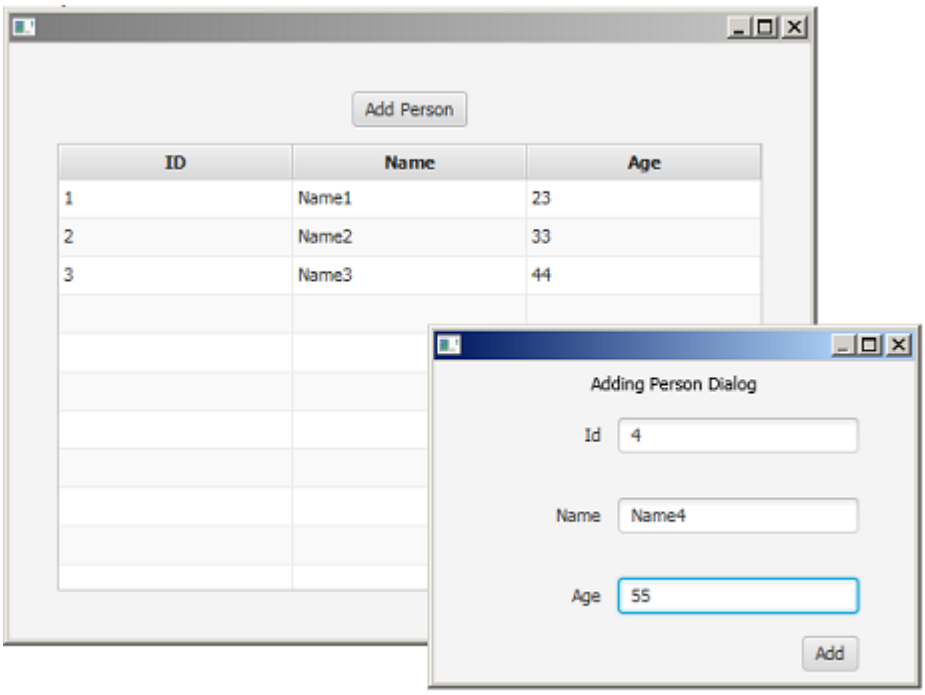
```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }

    public int getId() {
        return id.get();
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```

Capture d'écran



Lire les fenêtres en ligne: https://riptutorial.com/fr/javafx/topic/1496/les-fenetres

Chapitre 14: Liaisons JavaFX

Examples

Liaison de propriété simple

JavaFX dispose d'une API de liaison, qui permet de lier une propriété à l'autre. Cela signifie que chaque fois que la valeur d'une propriété est modifiée, la valeur de la propriété liée est automatiquement mise à jour. Un exemple de liaison simple:

```
SimpleIntegerProperty first =new SimpleIntegerProperty(5); //create a property with value=5
SimpleIntegerProperty second=new SimpleIntegerProperty();

public void test()
{
    System.out.println(second.get()); // '0'
    second.bind(first);               //bind second property to first
    System.out.println(second.get()); // '5'
    first.set(16);                    //set first property's value
    System.out.println(second.get()); // '16' - the value was automatically updated
}
```

Vous pouvez également lier une propriété primitive en appliquant une addition, une soustraction, une division, etc.:

```
public void test2()
{
        second.bind(first.add(100));
        System.out.println(second.get()); //'105'
        second.bind(first.subtract(50));
        System.out.println(second.get()); //'-45'
}
```

Tout objet peut être placé dans SimpleObjectProperty:

```
SimpleObjectProperty<Color> color=new SimpleObjectProperty<>(Color.web("45f3d1"));
```

Il est possible de créer des liaisons bidirectionnelles. Dans ce cas, les propriétés dépendent les unes des autres.

```
public void test3()
{
        second.bindBidirectional(first);
        System.out.println(second.get()+" "+first.get());
        second.set(1000);
        System.out.println(second.get()+" "+first.get()); //both are '1000'
}
```

Lire Liaisons JavaFX en ligne: https://riptutorial.com/fr/javafx/topic/7014/liaisons-javafx

Chapitre 15: Mises en page

Examples

StackPane

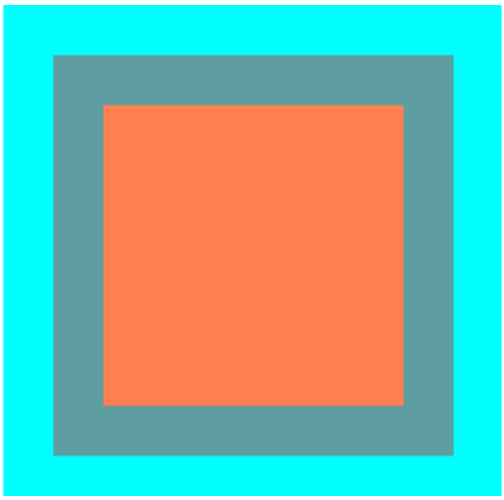
`StackPane` dispose ses enfants dans une pile dos à face.

L'ordre des enfants est défini par l'ordre de la liste des enfants (accessible en appelant `getChildren`): le 0ème enfant est le dernier et le dernier enfant au sommet de la pile.

Le stackpane tente de redimensionner chaque enfant pour remplir sa propre zone de contenu. Dans le cas où un enfant ne peut pas être redimensionné pour remplir la zone du `StackPane` (soit parce qu'il ne peut pas être redimensionné ou que sa taille maximale est empêchée), il sera alors aligné dans la zone avec la propriété `alignmentProperty` du tableau, par défaut `Pos.CENTER`.

Exemple

```
// Create a StackPane
StackPane pane = new StackPane();

// Create three squares
Rectangle rectBottom = new Rectangle(250, 250);
rectBottom.setFill(Color.AQUA);
Rectangle rectMiddle = new Rectangle(200, 200);
rectMiddle.setFill(Color.CADETBLUE);
Rectangle rectUpper = new Rectangle(150, 150);
rectUpper.setFill(Color.CORAL);

// Place them on top of each other
pane.getChildren().addAll(rectBottom, rectMiddle, rectUpper);
```



HBox et VBox

Les dispositions `HBox` et `VBox` sont très similaires, les deux `VBox` leurs enfants sur une seule ligne.

Caractéristiques communes

Si une `HBox` ou une `VBox` ont une bordure et / ou un ensemble de remplissage, le contenu sera mis en forme dans ces encarts.

Ils répartissent chaque enfant géré indépendamment de la valeur de propriété visible de l'enfant; les enfants non gérés sont ignorés.

L'alignement du contenu est contrôlé par la propriété alignment, qui par défaut est **`Pos.TOP_LEFT`** .

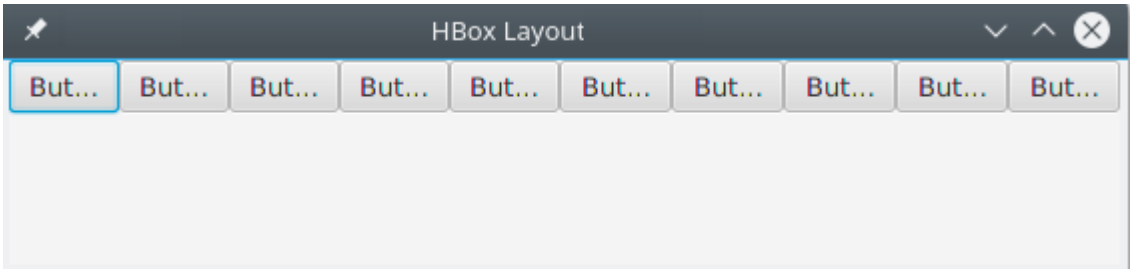
HBox

`HBox` dispose ses enfants sur une seule rangée horizontale de gauche à droite.

`HBox` redimensionnera les enfants (si redimensionnables) **à leur largeur** s **préférée** et utilisera sa propriété fillHeight pour déterminer s'il convient de redimensionner leurs hauteurs afin de remplir leur propre hauteur ou de garder leurs hauteurs à leur préférence (fillHeight est défini par défaut sur true).

Créer un HBox

```
// HBox example
HBox row = new HBox();
Label first = new Label("First");
Label second = new Label("Second");
row.getChildren().addAll(first, second);
```



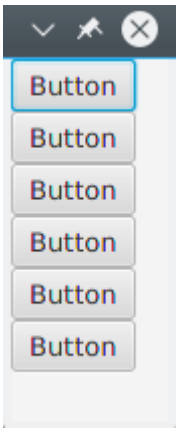
VBox

`VBox` dispose ses enfants dans une seule colonne verticale de haut en bas.

`VBox` redimensionnera les enfants (si redimensionnables) **à leur hauteur préférée** et utilisera sa propriété fillWidth pour déterminer s'il convient de redimensionner leurs largeurs afin de remplir sa propre largeur ou de conserver leurs largeurs à leur préférence (fillWidth par défaut à true).

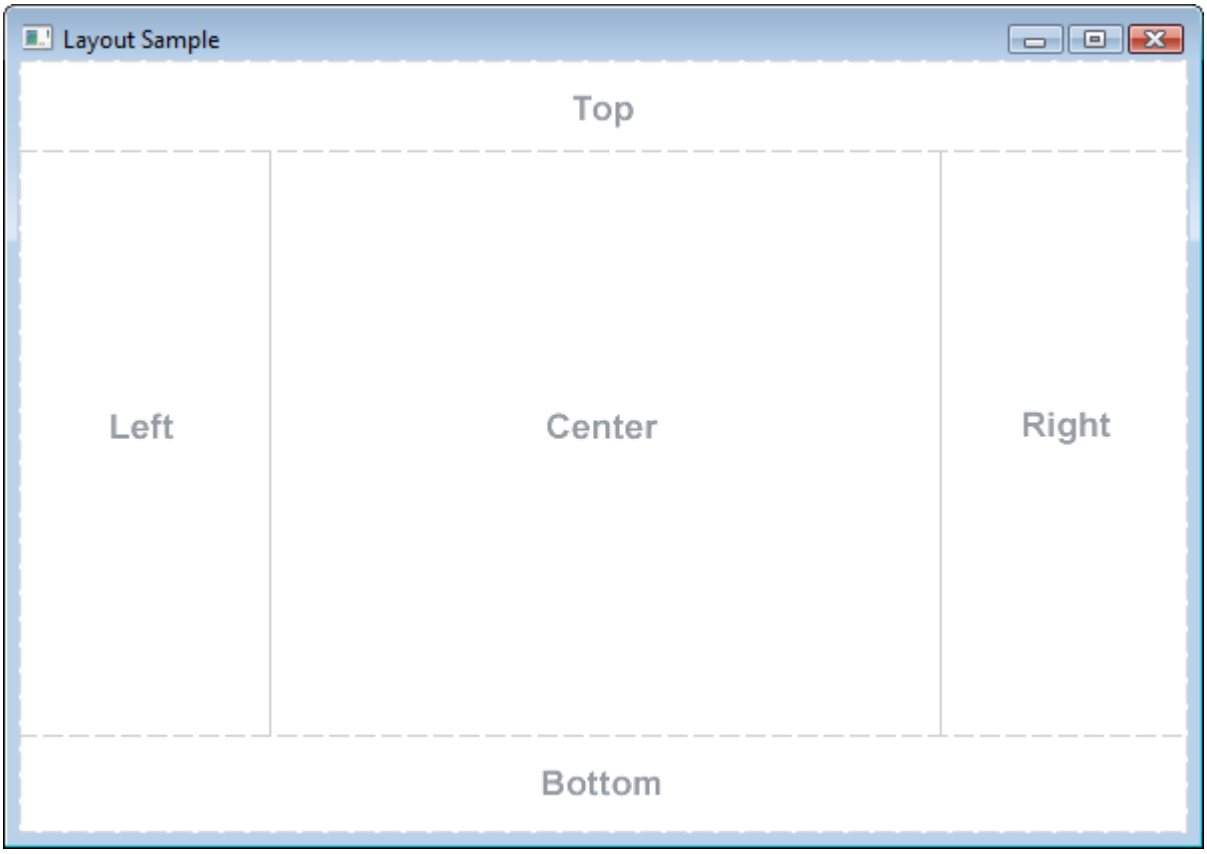
Créer un VBox

```
// VBox example
VBox column = new VBox();
Label upper = new Label("Upper");
Label lower = new Label("Lower");
column.getChildren().addAll(upper, lower);
```



BorderPane

Le `BorderPane` est `BorderPane` en cinq zones différentes.



Les zones de bordure (`Top` , `Right` , `Bottom` , `Left`) ont leur taille préférée en fonction de leur contenu. Par défaut, ils ne prendront que ce dont ils ont besoin, tandis que la zone `Center` prendra tout espace restant. Lorsque les zones frontalières sont vides, elles ne prennent pas de place.

Chaque zone ne peut contenir qu'un seul élément. Il peut être ajouté à l'aide des méthodes `setTop(Node)` , `setRight(Node)` , `setBottom(Node)` , `setLeft(Node)` , `setCenter(Node)` . Vous pouvez utiliser d'autres mises en page pour placer plus d'un élément dans une seule zone.

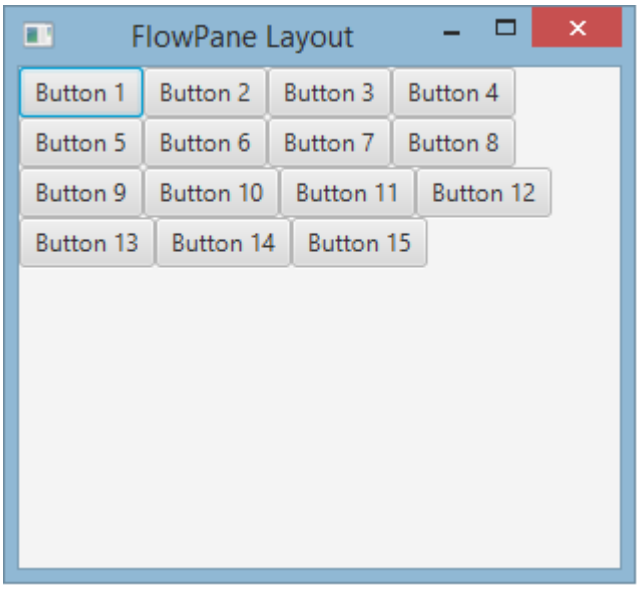
```
//BorderPane example
BorderPane pane = new BorderPane();

Label top = new Label("Top");
```

```
Label right = new Label("Right");

HBox bottom = new HBox();
bottom.getChildren().addAll(new Label("First"), new Label("Second"));

VBox left = new VBox();
left.getChildren().addAll(new Label("Upper"), new Label("Lower"));

StackPane center = new StackPane();
center.getChildren().addAll(new Label("Lorem"), new Label("ipsum"));

pane.setTop(top);        //The text "Top"
pane.setRight(right);    //The text "Right"
pane.setBottom(bottom);  //Row of two texts
pane.setLeft(left);      //Column of two texts
pane.setCenter(center);  //Two texts on each other
```

FlowPane

`FlowPane` dispose des nœuds en lignes ou en colonnes en fonction de l'espace horizontal ou vertical disponible. Il encapsule les nœuds sur la ligne suivante lorsque l'espace horizontal est inférieur au total de toutes les largeurs des nœuds; il encapsule les nœuds dans la colonne suivante lorsque l'espace vertical est inférieur au total de toutes les hauteurs des nœuds. Cet exemple illustre la disposition horizontale par défaut:

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
        FlowPane root = new FlowPane();
        for (int i=1; i<=15; i++) {
            Button b1=new Button("Button "+String.valueOf(i));
            root.getChildren().add(b1); //for adding button to root
        }
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("FlowPane Layout");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Constructeur `FlowPane` par défaut:

```
FlowPane root = new FlowPane();
```

Constructeurs `FlowPane` supplémentaires:

```
FlowPane() //Creates a horizontal FlowPane layout with hgap/vgap = 0 by default.
FlowPane(double hgap, double gap) //Creates a horizontal FlowPane layout with the specified
hgap/vgap.
FlowPane(double hgap, double vgap, Node... children) //Creates a horizontal FlowPane layout
with the specified hgap/vgap.
FlowPane(Node... children) //Creates a horizontal FlowPane layout with hgap/vgap = 0.
FlowPane(Orientation orientation) //Creates a FlowPane layout with the specified orientation
and hgap/vgap = 0.
FlowPane(Orientation orientation, double hgap, double gap) //Creates a FlowPane layout with
the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, double hgap, double vgap, Node... children) //Creates a
FlowPane layout with the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, Node... children) //Creates a FlowPane layout with the
specified orientation and hgap/vgap = 0.
```

L'ajout de nœuds à la disposition utilise les méthodes `add()` ou `addAll()` du `Pane` parent:

```
Button btn = new Button("Demo Button");
root.getChildren().add(btn);
root.getChildren().addAll(…);
```

Par défaut, un `FlowPane` dispose de nœuds enfants de gauche à droite. Pour modifier l'alignement du flux, appelez la méthode `setAlignment()` en transmettant une valeur énumérée de type `Pos` .

Quelques alignements de flux couramment utilisés:

```
root.setAlignment(Pos.TOP_RIGHT);     //for top right
root.setAlignment(Pos.TOP_CENTER);    //for top Center
root.setAlignment(Pos.CENTER);        //for Center
root.setAlignment(Pos.BOTTOM_RIGHT);  //for bottom right
```

`GridPane` dispose ses enfants dans une grille flexible de lignes et de colonnes.

Enfants de la grillePane

Un enfant peut être placé n'importe où dans le `GridPane` et peut s'étendre sur plusieurs lignes / colonnes (la portée par défaut est 1) et son placement dans la grille est défini par ses contraintes de disposition:

Contrainte	La description
columnIndex	colonne où la zone de mise en page de l'enfant commence.
rowIndex	rangée où la zone de mise en page de l'enfant commence.
columnSpan	le nombre de colonnes que la zone de mise en page de l'enfant couvre horizontalement.
rowSpan	le nombre de lignes de la zone de mise en page de l'enfant s'étendent verticalement.

Le nombre total de lignes / colonnes n'a pas besoin d'être spécifié à l'avance, car la grille étendra / contractera automatiquement la grille afin de prendre en charge le contenu.

Ajouter des enfants au GridPane

Afin d'ajouter de nouveaux `Node` à un `GridPane` les **contraintes de mise en page** sur les enfants doivent être définies à l'aide de la méthode statique de la classe `GridPane` , puis ces enfants peuvent être ajoutés à une instance `GridPane` .

```
GridPane gridPane = new GridPane();

// Set the constraints: first row and first column
Label label = new Label("Example");
GridPane.setRowIndex(label, 0);
GridPane.setColumnIndex(label, 0);
// Add the child to the grid
gridpane.getChildren().add(label);
```

`GridPane` fournit des méthodes pratiques pour combiner ces étapes:

```
gridPane.add(new Button("Press me!"), 1, 0); // column=1 row=0
```

La classe `GridPane` fournit également des méthodes de réglage statiques pour définir le **rang et le colonne** des éléments enfants:

```
Label labelLong = new Label("Its a long text that should span several rows");
GridPane.setColumnSpan(labelLong, 2);
gridPane.add(labelLong, 0, 1);  // column=0 row=1
```

Taille des colonnes et des rangées

Par défaut, les lignes et les colonnes seront dimensionnées pour correspondre à leur contenu. En cas de besoin de **contrôle explicite des tailles de lignes et de colonnes** , les instances `RowConstraints` et `ColumnConstraints` peuvent être ajoutées au `GridPane` . L'ajout de ces deux contraintes redimensionnera l'exemple ci-dessus pour avoir la première colonne de 100 pixels, la deuxième colonne de 200 pixels.

```
gridPane.getColumnConstraints().add(new ColumnConstraints(100));
gridPane.getColumnConstraints().add(new ColumnConstraints(200));
```

Par défaut, `GridPane` redimensionne les lignes / colonnes à leur taille préférée, même si la taille du tableau est supérieure à sa taille préférée. Pour prendre en charge **les tailles de colonne / ligne dynamiques** , les deux classes de contstaints fournissent trois propriétés: taille minimale, taille maximale et taille préférée.

De plus, `ColumnConstraints` fournit `setHGrow` et `RowConstraints` fournit des méthodes `setVGrow` pour **affecter la priorité de la croissance et de la réduction** . Les trois priorités prédéfinies sont:

- **Priorité** . TOUJOURS: Essayez toujours de croître (ou de réduire), en partageant l'augmentation (ou la diminution) de l'espace avec d'autres zones de mise en page qui ont toujours une croissance (ou une diminution).
- **Priority.SOMETIMES** : S'il n'y a pas d'autres zones de disposition avec l'agrandissement (ou la réduction) définies sur TOUJOURS ou si ces zones n'ont pas absorbé tout l'espace augmenté (ou diminué), alors partagera l'augmentation (ou la diminution) avec autre zone de mise en page de certains.
- **Priorité** . JAMAIS: la zone de mise en page ne grossira jamais (ou diminuera) quand il y aura une augmentation (ou une diminution) de l'espace disponible dans la région.

```
ColumnConstraints column1 = new ColumnConstraints(100, 100, 300);
column1.setHgrow(Priority.ALWAYS);
```

La colonne définie ci-dessus a une taille minimale de 100 pixels et elle essaiera toujours de croître jusqu'à atteindre sa largeur maximale de 300 pixels.

Il est également possible de définir un **dimensionnement en pourcentage** pour les lignes et les colonnes. L'exemple suivant définit un `GridPane` où la première colonne remplit 40% de la largeur de la grille, la seconde remplit les 60%.

```
GridPane gridpane = new GridPane();
ColumnConstraints column1 = new ColumnConstraints();
column1.setPercentWidth(40);
ColumnConstraints column2 = new ColumnConstraints();
column2.setPercentWidth(60);
```

```
gridpane.getColumnConstraints().addAll(column1, column2);
```

Alignement des éléments à l'intérieur des cellules de la grille

L'alignement des `Node` peut être défini à l'aide de la `setHalignment` (horizontal) de la classe `ColumnConstraints` et de la `setValignment` (vertical) de la classe `RowConstraints` .

```
ColumnConstraints column1 = new ColumnConstraints();
column1.setHalignment(HPos.RIGHT);

RowConstraints row1 = new RowConstraints();
row1.setValignment(VPos.CENTER);
```

TilePane

La disposition du volet de mosaïque est similaire à celle de FlowPane. TilePane place tous les nœuds dans une grille dans laquelle chaque cellule ou mosaïque a la même taille. Il organise les nœuds en lignes et colonnes ordonnées, horizontalement ou verticalement.

```
    import javafx.application.Application;
    import javafx.scene.Scene;
    import javafx.scene.control.Button;
    import javafx.scene.layout.TilePane;
    import javafx.stage.Stage;

    public class Main extends Application {

        @Override
        public void start(Stage primaryStage) {
            primaryStage.setTitle("TilePane Demo");
            double width = 400;
            double height = 300;
            TilePane root = new TilePane();
            root.setStyle("-fx-background-color:blue");
            // to set horizontal and vertical gap
            root.setHgap(20);
            root.setVgap(50);
            Button bl = new Button("Buttons");
            root.getChildren().add(bl);
            Button btn = new Button("Button");
            root.getChildren().add(btn);
            Button btn1 = new Button("Button 1");
            root.getChildren().add(btn1);
            Button btn2 = new Button("Button 2");
            root.getChildren().add(btn2);
            Button btn3 = new Button("Button 3");
            root.getChildren().add(btn3);
            Button btn4 = new Button("Button 4");
            root.getChildren().add(btn4);

            Scene scene = new Scene(root, width, height);
            primaryStage.setScene(scene);
            primaryStage.show();
        }
```

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

sortie

Pour créer Tilepane

```
TilePane root = new TilePane();
```

La méthode setHgap () et setVgap () est utilisée pour créer un espace entre la colonne et la colonne. nous pouvons également définir les colonnes pour la mise en page en utilisant

```
int columnCount = 2;
root.setPrefColumns(columnCount);
```

AnchorPane

`AnchorPane` a est une mise en page qui permet de placer le contenu à une distance spécifique de ses côtés.

Il existe 4 méthodes de réglage et 4 méthodes pour obtenir les distances dans `AnchorPane` . Le premier paramètre de ces méthodes est le `Node` enfant. Le second paramètre des paramètres est la valeur `Double` à utiliser. Cette valeur peut être `null` ce qui ne signifie aucune contrainte pour le côté donné.

méthode setter	méthode getter
setBottomAnchor	getBottomAnchor

méthode setter	méthode getter
setLeftAnchor	getLeftAnchor
setRightAnchor	getRightAnchor
setTopAnchor	getTopAnchor

Dans l'exemple suivant, placez des nœuds à des distances spécifiées des côtés.

La région `center` est également redimensionnée pour conserver les distances spécifiées par rapport aux côtés. Observez le comportement lorsque la fenêtre est redimensionnée.

```
public static void setBackgroundColor(Region region, Color color) {
    // change to 50% opacity
    color = color.deriveColor(0, 1, 1, 0.5);
    region.setBackground(new Background(new BackgroundFill(color, CornerRadii.EMPTY,
Insets.EMPTY)));
}

@Override
public void start(Stage primaryStage) {
    Region right = new Region();
    Region top = new Region();
    Region left = new Region();
    Region bottom = new Region();
    Region center = new Region();

    right.setPrefSize(50, 150);
    top.setPrefSize(150, 50);
    left.setPrefSize(50, 150);
    bottom.setPrefSize(150, 50);

    // fill with different half-transparent colors
    setBackgroundColor(right, Color.RED);
    setBackgroundColor(left, Color.LIME);
    setBackgroundColor(top, Color.BLUE);
    setBackgroundColor(bottom, Color.YELLOW);
    setBackgroundColor(center, Color.BLACK);

    // set distances to sides
    AnchorPane.setBottomAnchor(bottom, 50d);
    AnchorPane.setTopAnchor(top, 50d);
    AnchorPane.setLeftAnchor(left, 50d);
    AnchorPane.setRightAnchor(right, 50d);

    AnchorPane.setBottomAnchor(center, 50d);
    AnchorPane.setTopAnchor(center, 50d);
    AnchorPane.setLeftAnchor(center, 50d);
    AnchorPane.setRightAnchor(center, 50d);

    // create AnchorPane with specified children
    AnchorPane anchorPane = new AnchorPane(left, top, right, bottom, center);

    Scene scene = new Scene(anchorPane, 200, 200);

    primaryStage.setScene(scene);
    primaryStage.show();
```

```
}
```

Lire Mises en page en ligne: https://riptutorial.com/fr/javafx/topic/2121/mises-en-page

Chapitre 16: Pagination

Examples

Créer une pagination

Les paginations dans JavaFX utilisent un rappel pour obtenir les pages utilisées dans l'animation.

```
Pagination p = new Pagination();
p.setPageFactory(param -> new Button(param.toString()));
```

Cela crée une liste infinie de boutons numérotés `0..` puisque le constructeur zéro arg crée une pagination infinie. `setPageFactory` prend un rappel qui prend un int et retourne le noeud que nous voulons à cet index.

Avance automatique

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
fiveSecondsWonder.play();

stage.setScene(new Scene(p));
stage.show();
```

Cela fait avancer la pagination toutes les 5 secondes.

Comment ça marche

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
```

`fiveSecondsWonder` est un scénario qui déclenche un événement à chaque fois qu'il termine un cycle. Dans ce cas, le temps de cycle est de 5 secondes.

```
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
```

Cochez la pagination.

```
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
```

Définissez la chronologie pour qu'elle s'exécute pour toujours.

```
fiveSecondsWonder.play();
```

Créer une pagination d'images

```
ArrayList<String> images = new ArrayList<>();
images.add("some\\cool\\image");
images.add("some\\other\\cool\\image");
images.add("some\\cooler\\image");

Pagination p = new Pagination(3);
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Notez que les chemins doivent être des URL, pas des chemins de système de fichiers.

Comment ça marche

```
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Tout le reste n'est que du fluff, c'est là que se passe le vrai travail. `setPageFactory` prend un rappel qui prend un int et retourne le noeud que nous voulons à cet index. La première page correspond au premier élément de la liste, le second au deuxième élément de la liste, etc.

Lire Pagination en ligne: https://riptutorial.com/fr/javafx/topic/5165/pagination

Chapitre 17: Propriétés et observable

Remarques

Les propriétés sont observables et des écouteurs peuvent y être ajoutés. Ils sont systématiquement utilisés pour les propriétés des `Node` .

Examples

Types de propriétés et nom

Propriétés standard

Selon le type de propriété, il existe jusqu'à 3 méthodes pour une seule propriété. Soit `<property>` le nom d'une propriété et `<Property>` le nom de la propriété avec une première lettre en majuscule. Et laissez `T` être le type de la propriété; Pour les wrappers primitifs, nous utilisons le type primitif ici, par exemple, `String` pour `StringProperty` et `double` pour `ReadOnlyDoubleProperty` .

Nom de la méthode	Paramètres	Type de retour	Objectif
`<property>Property`	`()`	La propriété elle-même, par exemple `DoubleProperty` , `ReadOnlyStringProperty` , `ObjectProperty<VPos>`	renvoyer la propriété elle-même pour ajouter des écouteurs / des liaisons
`get<Property>`	`()`	`T`	renvoyer la valeur enveloppée dans la propriété
`set<Property>`	`(T)`	`void`	définir la valeur de la propriété

Notez que le setter n'existe pas pour les propriétés en lecture seule.

Propriétés de la liste en lecture seule

Les propriétés de liste en lecture seule sont des propriétés qui fournissent uniquement une méthode de lecture. Le type d'une telle propriété est `ObservableList` , de préférence avec un agrument de type spécifié. La valeur de cette propriété ne change jamais; le contenu de `ObservableList` peut être modifié à la place.

Propriétés de la carte en lecture seule

Semblable aux propriétés de liste en lecture seule, les propriétés de la carte en lecture seule fournissent uniquement un getter et le contenu peut être modifié au lieu de la valeur de la propriété. Le getter renvoie un `ObservableMap` .

Exemple de StringProperty

L'exemple suivant montre la déclaration d'une propriété (`StringProperty` dans ce cas) et montre comment lui ajouter un `ChangeListener` .

```
import java.text.MessageFormat;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Person {

    private final StringProperty name = new SimpleStringProperty();

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.nameProperty().addListener(new ChangeListener<String>() {

            @Override
            public void changed(ObservableValue<? extends String> observable, String oldValue,
String newValue) {
                System.out.println(MessageFormat.format("The name changed from \"{0}\" to
\"{1}\"", oldValue, newValue));
            }

        });

        person.setName("Anakin Skywalker");
        person.setName("Darth Vader");
    }

}
```

Exemple ReadOnlyIntegerProperty

Cet exemple montre comment utiliser une propriété wrapper en lecture seule pour créer une propriété dans laquelle il est impossible d'écrire. Dans ce cas, le `cost` et le `price` peuvent être modifiés, mais le `profit` sera toujours le `price - cost` .

```
import java.text.MessageFormat;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ReadOnlyIntegerProperty;
import javafx.beans.property.ReadOnlyIntegerWrapper;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Product {

    private final IntegerProperty price = new SimpleIntegerProperty();
    private final IntegerProperty cost = new SimpleIntegerProperty();
    private final ReadOnlyIntegerWrapper profit = new ReadOnlyIntegerWrapper();

    public Product() {
        // the property itself can be written to
        profit.bind(price.subtract(cost));
    }

    public final int getCost() {
        return this.cost.get();
    }

    public final void setCost(int value) {
        this.cost.set(value);
    }

    public final IntegerProperty costProperty() {
        return this.cost;
    }

    public final int getPrice() {
        return this.price.get();
    }

    public final void setPrice(int value) {
        this.price.set(value);
    }

    public final IntegerProperty priceProperty() {
        return this.price;
    }

    public final int getProfit() {
        return this.profit.get();
    }

    public final ReadOnlyIntegerProperty profitProperty() {
        // return a readonly view of the property
        return this.profit.getReadOnlyProperty();
    }

    public static void main(String[] args) {
        Product product = new Product();
        product.profitProperty().addListener(new ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> observable, Number oldValue,
Number newValue) {
                System.out.println(MessageFormat.format("The profit changed from {0}$ to
{1}$", oldValue, newValue));
```

```
            }

        });
        product.setCost(40);
        product.setPrice(50);
        product.setCost(20);
        product.setPrice(30);
    }

}
```

Lire Propriétés et observable en ligne: https://riptutorial.com/fr/javafx/topic/4436/proprietes-et-observable

Chapitre 18: ScrollPane

Introduction

Le ScrollPane est un contrôle qui offre une vue dynamique de son contenu. Cette vue est contrôlée de différentes manières; (bouton d'incrémentation / molette de la souris) pour avoir une vue intégrale du contenu.

Examples

A) Taille du contenu fixe:

La taille du contenu sera la même que celle de son conteneur ScrollPane.

```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane(content);  //Initialize and add content as a parameter
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane

scrollpane.setFitToWidth(true);  //Adapt the content to the width of ScrollPane
scrollpane.setFitToHeight(true); //Adapt the content to the height of ScrollPane


scrollpane.setHbarPolicy(ScrollBarPolicy.ALWAYS);  //Control the visibility of the Horizontal
ScrollBar
scrollpane.setVbarPolicy(ScrollBarPolicy.NEVER);  //Control the visibility of the Vertical
ScrollBar
//There are three types of visibility (ALWAYS/AS_NEEDED/NEVER)
```

B) Taille du contenu dynamique:

La taille du contenu changera en fonction des éléments ajoutés qui dépassent les limites de contenu dans les deux axes (horizontal et vertical) visibles en se déplaçant dans la vue.

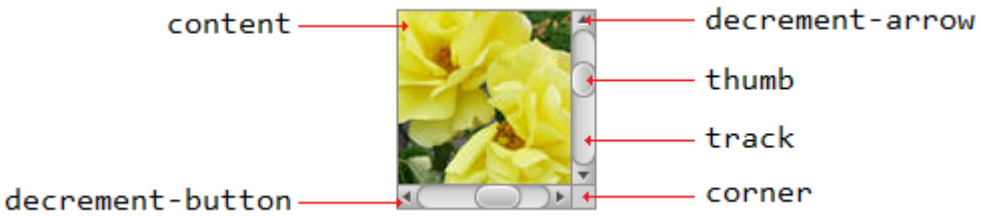
```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane();
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane
content.setMinSize(300,300); //Here a minimum size is set so that the container can be
extended.
scrollpane.setContent(content); // we add the content to the ScrollPane
```

Note: Ici, nous n'avons pas besoin des deux méthodes (setFitToWidth / setFitToHeight).

Styling the ScrollPane:

L'apparence du ScrollPane peut être facilement modifiée, en ayant certaines notions de " *CSS* " et en respectant certaines " *propriétés* " de contrôle et en ayant bien sûr une certaine " *imagination* ".

A) Les éléments qui composent ScrollPane:



B) Propriétés CSS:

```
.scroll-bar:vertical .track{}

.scroll-bar:horizontal .track{}

.scroll-bar:horizontal .thumb{}

.scroll-bar:vertical .thumb{}


.scroll-bar:vertical *.increment-button,
.scroll-bar:vertical *.decrement-button{}

.scroll-bar:vertical *.increment-arrow .content,
.scroll-bar:vertical *.decrement-arrow .content{}

.scroll-bar:vertical *.increment-arrow,
.scroll-bar:vertical *.decrement-arrow{}


.scroll-bar:horizontal *.increment-button,
.scroll-bar:horizontal *.decrement-button{}

.scroll-bar:horizontal *.increment-arrow .content,
.scroll-bar:horizontal *.decrement-arrow .content{}

.scroll-bar:horizontal *.increment-arrow,
.scroll-bar:horizontal *.decrement-arrow{}

.scroll-pane .corner{}

.scroll-pane{}
```

Lire ScrollPane en ligne: https://riptutorial.com/fr/javafx/topic/8259/scrollpane

Chapitre 19: TableView

Examples

Sample TableView avec 2 colonnes

Article de table

La classe suivante contient 2 propriétés un nom (`String`) et la taille (`double`). Les deux propriétés sont encapsulées dans les propriétés JavaFX pour permettre à `TableView` d'observer les modifications.

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public Person(String name, double size) {
        this.size = new SimpleDoubleProperty(this, "size", size);
        this.name = new SimpleStringProperty(this, "name", name);
    }

    private final StringProperty name;
    private final DoubleProperty size;

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public final double getSize() {
        return this.size.get();
    }

    public final void setSize(double value) {
        this.size.set(value);
    }

    public final DoubleProperty sizeProperty() {
        return this.size;
    }

}
```

Exemple d'application

Cette application montre une `TableView` avec 2 colonnes; un pour le nom et un pour la taille d'une `Person` . La sélection de l'une des `Person` ajoute les données à `TextField` dessous de `TableView` et permet à l'utilisateur de modifier les données. Notez qu'une fois la modification validée, la `TableView` est automatiquement mise à jour.

Pour chaque `TableColumn` ajoutée à la `TableView` une `cellValueFactory` est affectée. Cette fabrique est chargée de convertir les éléments de table (`Person` s) en `ObservableValue` s contenant la valeur à afficher dans la cellule de table et permettant à `TableView` d'écouter toute modification de cette valeur.

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.TextFormatter;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;
import javafx.util.StringConverter;

public class TableSample extends Application {

    @Override
    public void start(Stage primaryStage) {
        // data for the tableview. modifying this list automatically updates the tableview
        ObservableList<Person> data = FXCollections.observableArrayList(
                new Person("John Doe", 1.75),
                new Person("Mary Miller", 1.70),
                new Person("Frank Smith", 1.80),
                new Person("Charlotte Hoffman", 1.80)
        );

        TableView<Person> tableView = new TableView<>(data);

        // table column for the name of the person
        TableColumn<Person, String> nameColumn = new TableColumn<>("Name");
        nameColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
String>, ObservableValue<String>>() {

            @Override
            public ObservableValue<String> call(TableColumn.CellDataFeatures<Person, String>
param) {
                return param.getValue().nameProperty();
            }
        });

        // column for the size of the person
        TableColumn<Person, Number> sizeColumn = new TableColumn<>("Size");
```

```
        sizeColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
Number>, ObservableValue<Number>>() {

            @Override
            public ObservableValue<Number> call(TableColumn.CellDataFeatures<Person, Number>
param) {
                return param.getValue().sizeProperty();
            }
        });

        // add columns to tableview
        tableView.getColumns().addAll(nameColumn, sizeColumn);

        TextField name = new TextField();

        TextField size = new TextField();

        // convert input from textfield to double
        TextFormatter<Double> sizeFormatter = new TextFormatter<Double>(new
StringConverter<Double>() {

            @Override
            public String toString(Double object) {
                return object == null ? "" : object.toString();
            }

            @Override
            public Double fromString(String string) {
                if (string == null || string.isEmpty()) {
                    return null;
                } else {
                    try {
                        double val = Double.parseDouble(string);
                        return val < 0 ? null : val;
                    } catch (NumberFormatException ex) {
                        return null;
                    }
                }
            }

        });
        size.setTextFormatter(sizeFormatter);

        Button commit = new Button("Change Item");
        commit.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                Person p = tableView.getSelectionModel().getSelectedItem();
                p.setName(name.getText());
                Double value = sizeFormatter.getValue();
                p.setSize(value == null ? -1d : value);
            }

        });

        // listen for changes in the selection to update the data in the textfields
        tableView.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Person>() {

            @Override
```

```
            public void changed(ObservableValue<? extends Person> observable, Person oldValue,
Person newValue) {
                commit.setDisable(newValue == null);
                if (newValue != null) {
                    sizeFormatter.setValue(newValue.getSize());
                    name.setText(newValue.getName());
                }
            }

        });

        HBox editors = new HBox(5, new Label("Name:"), name, new Label("Size: "), size,
commit);

        VBox root = new VBox(10, tableView, editors);

        Scene scene = new Scene(root);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

PropertyValueFactory

`PropertyValueFactory` peut être utilisé comme `cellValueFactory` dans une `TableColumn` . Il utilise la réflexion pour accéder aux méthodes correspondant à un certain modèle pour récupérer les données d'un élément `TableView` :

Exemple

```
TableColumn<Person, String> nameColumn = ...
PropertyValueFactory<Person, String> valueFactory = new PropertyValueFactory<>("name");
nameColumn.setCellValueFactory(valueFactory);
```

Le nom de la méthode utilisée pour obtenir les données dépend du paramètre de constructeur pour `PropertyValueFactory` .

- **Méthode de propriété:** Ce type de méthode est censé renvoyer une valeur `ObservableValue` contenant les données. Des changements peuvent être observés. Ils doivent correspondre au modèle `<constructor parameter>Property` et ne prendre aucun paramètre.
- **Méthode Getter:** Ce type de méthode s'attend à renvoyer directement la valeur (`String` dans l'exemple ci-dessus). Le nom de la méthode doit correspondre au modèle `get<Constructor parameter>` . Notez que le `<Constructor parameter>` commence ici par une *lettre majuscule* . Cette méthode ne doit pas prendre de paramètres.

Exemples de noms de méthodes

paramètre constructeur (sans guillemets)	nom de la méthode de propriété	nom de la méthode getter
foo	fooProperty	getFoo
fooBar	fooBarProperty	getFooBar
XYZ	XYZProperty	getXYZ
listeIndex	listIndexProperty	getListIndex
une valeur	aValueProperty	getAValue

Personnalisation du look de TableCell en fonction de l'article

Parfois, une colonne doit afficher un contenu différent de la valeur de l'élément `toString` . Dans ce cas, la `TableCell` créée par la `cellFactory` de la `TableColumn` est personnalisée pour modifier la disposition en fonction de l'élément.

Remarque importante: `TableView` crée uniquement les `TableCell` qui sont affichées dans l'interface utilisateur. Les éléments à l'intérieur des cellules peuvent changer et même devenir vides. Le programmeur doit veiller à annuler toutes les modifications apportées à la `TableCell` lorsqu'un élément a été ajouté lorsqu'il a été supprimé. Sinon, le contenu peut encore être affiché dans une cellule où "il n'appartient pas".

Dans l'exemple ci-dessous, la définition d'un élément entraîne la définition du texte et de l'image affichée dans `ImageView` :

```
image.setImage(item.getEmoji());
setText(item.getValue());
```

Si l'élément devient `null` ou que la cellule devient vide, ces modifications sont annulées en définissant les valeurs sur `null` :

```
setText(null);
image.setImage(null);
```

L'exemple suivant montre un emoji en plus du texte dans un objet `TableCell` .

La méthode `updateItem` est appelée chaque fois que l'élément d'une `Cell` est modifié. En remplaçant cette méthode, vous pouvez réagir aux changements et ajuster l'apparence de la cellule. Ajouter un écouteur à `itemProperty()` d'une cellule serait une alternative, mais dans de nombreux cas, `TableCell` est étendu.

Type d'élément

```
import javafx.scene.image.Image;

// enum providing image and text for certain feelings
```

```
public enum Feeling {
    HAPPY("happy",
"https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Emojione_1F600.svg/64px-
Emojione_1F600.svg.png"),
    SAD("sad",
"https://upload.wikimedia.org/wikipedia/commons/thumb/4/42/Emojione_1F62D.svg/64px-
Emojione_1F62D.svg.png")
    ;
    private final Image emoji;
    private final String value;

    Feeling(String value, String url) {
        // load image in background
        emoji = new Image(url, true);
        this.value = value;
    }

    public Image getEmoji() {
        return emoji;
    }

    public String getValue() {
        return value;
    }

}
```

Code dans la classe Application

```
import javafx.application.Application;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

public class EmotionTable extends Application {

    public static class Item {

        private final ObjectProperty<Feeling> feeling;

        public Item(Feeling feeling) {
            this.feeling = new SimpleObjectProperty<>(feeling);
        }

        public final Feeling getFeeling() {
            return this.feeling.get();
```

```
        }

        public final void setFeeling(Feeling value) {
            this.feeling.set(value);
        }

        public final ObjectProperty<Feeling> feelingProperty() {
            return this.feeling;
        }

    }

    @Override
    public void start(Stage primaryStage) {
        TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD),
                null,
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD)
        ));

        EventHandler<ActionEvent> eventHandler = new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                // change table items depending on userdata of source
                Node source = (Node) event.getSource();
                Feeling targetFeeling = (Feeling) source.getUserData();
                for (Item item : table.getItems()) {
                    if (item != null) {
                        item.setFeeling(targetFeeling);
                    }
                }
            }

        };

        TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

        feelingColumn.setCellValueFactory(new PropertyValueFactory<>("feeling"));

        // use custom tablecell to display emoji image
        feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item,
Feeling>>() {

            @Override
            public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
                return new EmojiCell<>();
            }
        });

        table.getColumns().add(feelingColumn);

        Button sunshine = new Button("sunshine");
        Button rain = new Button("rain");

        sunshine.setOnAction(eventHandler);
```

```
        rain.setOnAction(eventHandler);

        sunshine.setUserData(Feeling.HAPPY);
        rain.setUserData(Feeling.SAD);

        Scene scene = new Scene(new VBox(10, table, new HBox(10, sunshine, rain)));

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Classe de cellule

```
import javafx.scene.control.TableCell;
import javafx.scene.image.ImageView;

public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {
        // add ImageView as graphic to display it in addition
        // to the text in the cell
        image = new ImageView();
        image.setFitWidth(64);
        image.setFitHeight(64);
        image.setPreserveRatio(true);

        setGraphic(image);
        setMinHeight(70);
    }

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

Ajouter un bouton à la tableview

Vous pouvez ajouter un bouton ou un autre composant javafx à Tableview en utilisant la méthode `setCellFactory(Callback value)` colonne.

Exemple d'application

Dans cette application, nous allons ajouter un bouton à TableView. Lorsque vous cliquez sur ce bouton de colonne, les données de la même ligne que le bouton sont sélectionnées et leurs informations sont imprimées.

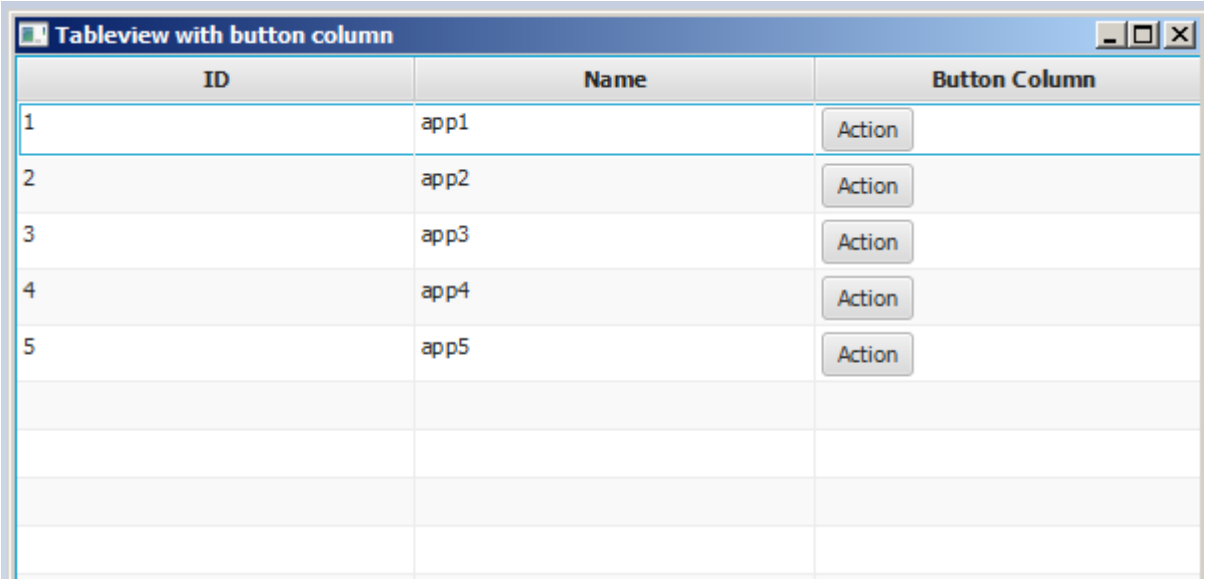
Dans la méthode `addButtonToTable()` , le rappel `cellFactory` est responsable de l'ajout du bouton à la colonne associée. Nous définissons le cellFactory appelable et implémentons sa méthode `call(...)` pour obtenir `TableCell` avec le bouton, puis cette `cellFactory` définit la `cellFactory` associée à la `setCellFactory(..)` . Dans notre exemple, il s'agit de `colBtn.setCellFactory(cellFactory)` . SSCCE est ci-dessous:

```
import javafx.application.Application;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import javafx.util.Callback;

public class TableViewSample extends Application {

    private final TableView<Data> table = new TableView<>();
    private final ObservableList<Data> tvObservableList = FXCollections.observableArrayList();

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

    @Override
    public void start(Stage stage) {

        stage.setTitle("Tableview with button column");
        stage.setWidth(600);
        stage.setHeight(600);

        setTableappearance();

        fillTableObservableListWithSampleData();
        table.setItems(tvObservableList);

        TableColumn<Data, Integer> colId = new TableColumn<>("ID");
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Data, String> colName = new TableColumn<>("Name");
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));

        table.getColumns().addAll(colId, colName);

        addButtonToTable();
```

```
        Scene scene = new Scene(new Group(table));

        stage.setScene(scene);
        stage.show();
    }

    private void setTableappearance() {
        table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        table.setPrefWidth(600);
        table.setPrefHeight(600);
    }

    private void fillTableObservableListWithSampleData() {

        tvObservableList.addAll(new Data(1, "app1"),
                                new Data(2, "app2"),
                                new Data(3, "app3"),
                                new Data(4, "app4"),
                                new Data(5, "app5"));
    }

    private void addButtonToTable() {
        TableColumn<Data, Void> colBtn = new TableColumn("Button Column");

        Callback<TableColumn<Data, Void>, TableCell<Data, Void>> cellFactory = new
Callback<TableColumn<Data, Void>, TableCell<Data, Void>>() {
            @Override
            public TableCell<Data, Void> call(final TableColumn<Data, Void> param) {
                final TableCell<Data, Void> cell = new TableCell<Data, Void>() {

                    private final Button btn = new Button("Action");

                    {
                        btn.setOnAction((ActionEvent event) -> {
                            Data data = getTableView().getItems().get(getIndex());
                            System.out.println("selectedData: " + data);
                        });
                    }

                    @Override
                    public void updateItem(Void item, boolean empty) {
                        super.updateItem(item, empty);
                        if (empty) {
                            setGraphic(null);
                        } else {
                            setGraphic(btn);
                        }
                    }
                };
                return cell;
            }
        };

        colBtn.setCellFactory(cellFactory);

        table.getColumns().add(colBtn);

    }

    public class Data {
```

```
        private int id;
        private String name;

        private Data(int id, String name) {
            this.id = id;
            this.name = name;
        }

        public int getId() {
            return id;
        }

        public void setId(int ID) {
            this.id = ID;
        }

        public String getName() {
            return name;
        }

        public void setName(String nme) {
            this.name = nme;
        }

        @Override
        public String toString() {
            return "id: " + id + " - " + "name: " + name;
        }

    }
}
```

Capture d'écran:



Lire TableView en ligne: https://riptutorial.com/fr/javafx/topic/2229/tableview

Chapitre 20: Toile

Introduction

Un `Canvas` est un `Node` JavaFX, représenté par une zone rectangulaire vierge pouvant afficher des images, des formes et du texte. Chaque `Canvas` contient exactement un objet `GraphicsContext` , chargé de recevoir et de mettre en mémoire tampon les appels de dessin qui, à la fin, sont rendus à l'écran par `Canvas` .

Examples

Formes de base

`GraphicsContext` fournit un ensemble de méthodes pour dessiner et remplir des formes géométriques. En règle générale, ces méthodes ont besoin de coordonnées comme paramètres, soit directement, soit sous la forme d'un tableau de valeurs `double` . Les coordonnées sont toujours relatives au `Canvas` , dont l'origine est située dans le coin supérieur gauche.

Remarque: `GraphicsContext` ne dessine pas en dehors des limites du `Canvas` , c'est-à-dire qu'essayer de dessiner en dehors de la zone de `Canvas` définie par sa taille et de le redimensionner ultérieurement ne donnera aucun résultat.

L'exemple ci-dessous montre comment dessiner trois formes géométriques remplies semi-transparentes avec un trait noir.

```
Canvas canvas = new Canvas(185, 70);
GraphicsContext gc = canvas.getGraphicsContext2D();

// Set stroke color, width, and global transparency
gc.setStroke(Color.BLACK);
gc.setLineWidth(2d);
gc.setGlobalAlpha(0.5d);

// Draw a square
gc.setFill(Color.RED);
gc.fillRect(10, 10, 50, 50);
gc.strokeRect(10, 10, 50, 50);

// Draw a triangle
gc.setFill(Color.GREEN);
gc.fillPolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);
gc.strokePolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);

// Draw a circle
gc.setFill(Color.BLUE);
gc.fillOval(130, 10, 50, 50);
gc.strokeOval(130, 10, 50, 50);
```

Lire Toile en ligne: https://riptutorial.com/fr/javafx/topic/8935/toile

Chapitre 21: WebView et WebEngine

Remarques

`WebView` est le noeud JavaFX intégré à l'arborescence des composants JavaFX. Il gère un `WebEngine` et affiche son contenu.

`WebEngine` est le moteur de `WebEngine` sous-jacent, qui fait tout le travail.

Examples

Chargement d'une page

```
WebView wv = new WebView();
WebEngine we = wv.getEngine();
we.load("https://stackoverflow.com");
```

`WebView` est le shell d'interface utilisateur autour du `WebEngine` . Presque tous les contrôles pour l'interaction sans interface utilisateur avec une page sont effectués via la classe `WebEngine` .

Obtenir l'historique des pages d'une WebView

```
WebHistory history = webView.getEngine().getHistory();
```

L'historique est essentiellement une liste d'entrées. Chaque entrée représente une page visitée et permet d'accéder aux informations de page pertinentes, telles que l'URL, le titre et la date de la dernière visite de la page.

La liste peut être obtenue en utilisant la méthode `getEntries()` . L'historique et la liste correspondante des entrées changent lorsque `WebEngine` navigue sur le Web. La liste peut être étendue ou réduite en fonction des actions du navigateur. Ces modifications peuvent être écoutées par l'API ObservableList que la liste expose.

L'index de l'entrée d'historique associée à la page actuellement visitée est représenté par `currentIndexProperty()` . L'index actuel peut être utilisé pour accéder à n'importe quelle entrée de l'historique en utilisant la méthode `go(int)` . Le `maxSizeProperty()` définit la taille maximale de l'historique, qui correspond à la taille de la liste d'historique.

Vous trouverez ci-dessous un exemple sur la façon d' obtenir et de traiter la liste des éléments de l'historique Web .

Un `ComboBox` (comboBox) est utilisé pour stocker les éléments de l'historique. En utilisant `ListChangeListener` sur le `WebHistory` le `ComboBox` est mis à jour vers `WebHistory` actuel. Sur le `ComboBox` y a un `EventHandler` qui redirige vers la page sélectionnée.

```
final WebHistory history = webEngine.getHistory();

comboBox.setItems(history.getEntries());
comboBox.setPrefWidth(60);
comboBox.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent ev) {
        int offset =
                comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    }
});

history.currentIndexProperty().addListener(new ChangeListener<Number>() {

    @Override
    public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number
newValue) {
        // update currently selected combobox item
        comboBox.getSelectionModel().select(newValue.intValue());
    }
});

// set converter for value shown in the combobox:
//   display the urls
comboBox.setConverter(new StringConverter<WebHistory.Entry>() {

    @Override
    public String toString(WebHistory.Entry object) {
        return object == null ? null : object.getUrl();
    }

    @Override
    public WebHistory.Entry fromString(String string) {
        throw new UnsupportedOperationException();
    }
});
```

envoyer des alertes Javascript de la page Web affichée au journal des applications Java.

```
private final Logger logger = Logger.getLogger(getClass().getCanonicalName());

WebView webView = new WebView();
webEngine = webView.getEngine();

webEngine.setOnAlert(event -> logger.warning(() -> "JS alert: " + event.getData()));
```

Communication entre l'application Java et Javascript dans la page Web

Lorsque vous utilisez une WebView pour afficher votre propre page Web personnalisée et que celle-ci contient du Javascript, il peut être nécessaire d'établir une communication bidirectionnelle entre le programme Java et le Javascript de la page Web.

Cet exemple montre comment configurer une telle communication.

La page Web doit afficher un champ de saisie et un bouton. En cliquant sur le bouton, la valeur du champ de saisie est envoyée à l'application Java, qui la traite. Après traitement, un résultat est envoyé à Javascript qui affiche à son tour le résultat sur la page Web.

Le principe de base est que pour la communication de Javascript vers Java, un objet est créé en Java et défini dans la page Web. Et pour l'autre sens, un objet est créé en Javascript et extrait de la page Web.

Le code suivant montre la partie Java, je l'ai gardé dans un seul fichier:

```
package com.sothawo.test;

import javafx.application.Application;
import javafx.concurrent.Worker;
import javafx.scene.Scene;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

import java.io.File;
import java.net.URL;

/**
 * @author P.J. Meisch (pj.meisch@sothawo.com).
 */
public class WebViewApplication extends Application {

    /** for communication to the Javascript engine. */
    private JSObject javascriptConnector;

    /** for communication from the Javascript engine. */
    private JavaConnector javaConnector = new JavaConnector();;

    @Override
    public void start(Stage primaryStage) throws Exception {
        URL url = new File("./js-sample.html").toURI().toURL();

        WebView webView = new WebView();
        final WebEngine webEngine = webView.getEngine();

        // set up the listener
        webEngine.getLoadWorker().stateProperty().addListener((observable, oldValue, newValue)
-> {
            if (Worker.State.SUCCEEDED == newValue) {
                // set an interface object named 'javaConnector' in the web engine's page
                JSObject window = (JSObject) webEngine.executeScript("window");
                window.setMember("javaConnector", javaConnector);

                // get the Javascript connector object.
                javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
            }
        });

        Scene scene = new Scene(webView, 300, 150);
        primaryStage.setScene(scene);
        primaryStage.show();

        // now load the page
```

```
        webEngine.load(url.toString());
    }

    public class JavaConnector {
        /**
         * called when the JS side wants a String to be converted.
         *
         * @param value
         *         the String to convert
         */
        public void toLowerCase(String value) {
            if (null != value) {
                javascriptConnector.call("showResult", value.toLowerCase());
            }
        }
    }
}
```

Lorsque la page a été chargée, un objet `JavaConnector` (défini par la classe interne et créé en tant que champ) est défini dans la page Web par ces appels:

```
JSObject window = (JSObject) webEngine.executeScript("window");
window.setMember("javaConnector", javaConnector);
```

L'objet `javascriptConnector` est extrait de la page Web avec

```
javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
```

Lorsque la méthode `toLowerCase(String)` de `JavaConnector` est appelée, la valeur transmise est convertie, puis renvoyée via l'objet `javascriptConnector` .

Et ceci est le code HTML et JavaScript:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Sample</title>
    </head>
    <body>
        <main>

            <div><input id="input" type="text"></div>
            <button onclick="sendToJava();">to lower case</button>
            <div id="result"></div>

        </main>

        <script type="text/javascript">
            function sendToJava () {
                var s = document.getElementById('input').value;
                javaConnector.toLowerCase(s);
            };

            var jsConnector = {
                showResult: function (result) {
```

```
                    document.getElementById('result').innerHTML = result;
                }
            };

            function getJsConnector() {
                return jsConnector;
            };
        </script>
    </body>
</html>
```

La fonction `sendToJava` appelle la méthode de `JavaConnector` définie par le code Java:

```
function sendToJava () {
    var s = document.getElementById('input').value;
    javaConnector.toLowerCase(s);
};
```

et la fonction appelée par le code Java pour récupérer le `javascriptConnector` renvoie simplement l'objet `jsConnector` :

```
var jsConnector = {
    showResult: function (result) {
        document.getElementById('result').innerHTML = result;
    }
};

function getJsConnector() {
    return jsConnector;
};
```

Le type d'argument des appels entre Java et Javascript n'est pas limité aux chaînes. Plus d'informations sur les types possibles et la conversion sont disponibles dans le doc de l'API JSObject .

Lire WebView et WebEngine en ligne: https://riptutorial.com/fr/javafx/topic/5156/webview-et-webengine

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec javafx	Community, CraftedCart, D3181, DVarga, fabian, Ganesh, Hendrik Ebbers, Petter Friberg
2	Animation	fabian, J Atkin
3	Bouton	Dth, DVarga, J Atkin, Maverick283, Nico T, Squidward
4	Bouton radio	Nico T
5	Créateur de scène	Ashlyn Campbell, José Pereda
6	CSS	fabian
7	Dialogues	fabian, GltknBtn, Modus Tollens
8	Filetage	Brendan, fabian, GOXR3PLUS, James_D, Koko Essam, sazzy4o
9	FXML et contrôleurs	D3181, fabian, James_D
10	Graphique	Dth, James_D, Jinu P C
11	Impression	J Atkin, Squidward
12	Internationalisation en JavaFX	ItachiUchiha, Joffrey, Nico T, P.J.Meisch
13	les fenêtres	fabian, GltknBtn
14	Liaisons JavaFX	Alexiy
15	Mises en page	DVarga, fabian, Filip Smola, Jinu P C, Sohan Chowdhury, trashgod
16	Pagination	fabian, J Atkin
17	Propriétés et observable	fabian
18	ScrollPane	Bo Halim
19	TableView	Bo Halim, fabian, GltknBtn
20	Toile	Dth

21	WebView et WebEngine	fabian, J Atkin, James_D, P.J.Meisch, Squidward