Programmation Avancée et Application

Interfaces graphiques et programmation événementielle avec JavaFX

Jean-Guy Mailly
jean-guy.mailly@u-paris.fr

LIPADE - Université de Paris
http://www.math-info.univ-paris5.fr/~jmailly/

Interfaces graphiques et programmation événementielle avec JavaFX

1. Introduction
2. Construire son interface
   - Les bases
   - JavaFX : concevoir des applications comme une pièce de théâtre
   - Des composants de l'application : Control et ses descendants
   - Positionner les éléments : les layout
3. Programmation événementielle
   - Bases de la programmation événementielle en JavaFX
   - Utilisation des classes internes pour la gestion des événements
4. Utilisation de FXML et Scene Builder

Introduction

Présentation de JavaFX

- Une API Java pour la construction d'interfaces graphiques (GUI)
  - Depuis Java 8
  - **Attention** : Ne pas utiliser les anciennes API de Java (AWT et Swing), qui ne sont plus maintenues
  - Il faudra installer le SDK de JavaFX en plus de celui de Java
  - Dans ce cours : JavaFX 11 (version *Long Time Support*), nécessite Java 11 ou $>$
- Permet de créer des fenêtres et d'y placer des éléments graphiques avec lesquels un utilisateur peut interagir (via la souris, le clavier, l'écran tactile,. . . ...)
  - Boutons cliquables
  - Zones d'entrée/d'affichage de texte
  - Images
  - Menus
  - ...
- Il existe des projets qui étendent JavaFX (nouveaux composants) : ControlsFX, JFxtras

Description statique d'une GUI JavaFX

Différentes méthodes pour placer les composants dans la fenêtre

- « Pur Java » : utilisation de méthodes Java pour initialiser les composants et les placer dans la fenêtre
- FXML : langage XML conçu pour décrire une GUI JavaFX
  1. Décrire la GUI dans un fichier XML
  2. Dans le code Java, indiquer au programme de charger le fichier XML pour initialiser la GUI
- Scene Builder : interface graphique qui permet de créer le fichier XML « visuellement »

Partie dynamique de la GUI

- Programmation événementielle : on indique à chaque composant ce qui doit se passer lors d'un événement
  - Événement : clic sur un bouton, déplacement de la souris, appui sur une touche du clavier,...
  - Chaque composant est associé à un gestionnaire qui applique une méthode donnée lors d'un événement donné
- Exemple (pseudo-code) :

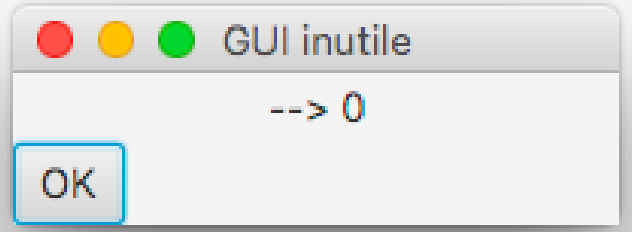
```
maMethode() {
   afficher("Hello World!")
}

class MonGestionnaireDeClics(){
   quandJeClique(){
      maMethode()
   }
}

Bouton b = new Bouton()
MonGestionnaireDeClics g = new MonGestionnaireDeClics()
b.ajouterGestionnaire(g)
```

Construire son interface

Premier exemple

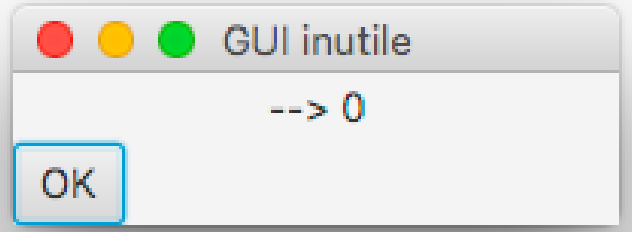
```
public class PremiereGUI extends Application {
    @Override
    public void start(Stage stage)
                                throws Exception {
        stage.setTitle("GUI inutile");
        BorderPane pane = new BorderPane();
        Label lab = new Label("--> 0");
        Button button = new Button("OK");
        pane.setCenter(lab);
        pane.setBottom(button);

        Scene scene = new Scene(pane);
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }
}
```

Premier exemple : résultat



- stage. setTitle ("GUI␣inutile") définit le titre de la fenêtre
- BorderPane pane = **new** BorderPane() crée un « panneau » de type BorderPane. Un panneau sert à contenir des éléments de la GUI
- Label lab = **new** Label("-->␣0") crée une zone d'affichage de texte
- Button button = **new** Button("OK") crée un bouton

Premier exemple : résultat



- pane.setCenter(lab) et pane.setBottom(button) placent le label et le bouton respectivement au centre et en bas du panneau
- Scene scene=**new** Scene(pane) crée une scène (niveau intermédiaire entre la fenêtre et les panneaux)
- stage.setScene(scene) place la scène dans la fenêtre
- stage.sizeToScene() définit la taille de la fenêtre à celle de la scène
- stage.show() affiche la fenêtre

Premier exemple complété

```
public class PremiereGUI extends Application {
    @Override
    public void start(Stage stage)
                                throws Exception {
        // Meme chose qu'avant...
    }

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

- Toute application JavaFX fait appel à launch dans sa méthode main
- launch est une méthode statique de Application :
  - crée l'objet Stage correspondant à la fenêtre principale
  - crée une instance de PremiereGUI
  - applique la méthode start sur cette instance

Un code modulaire

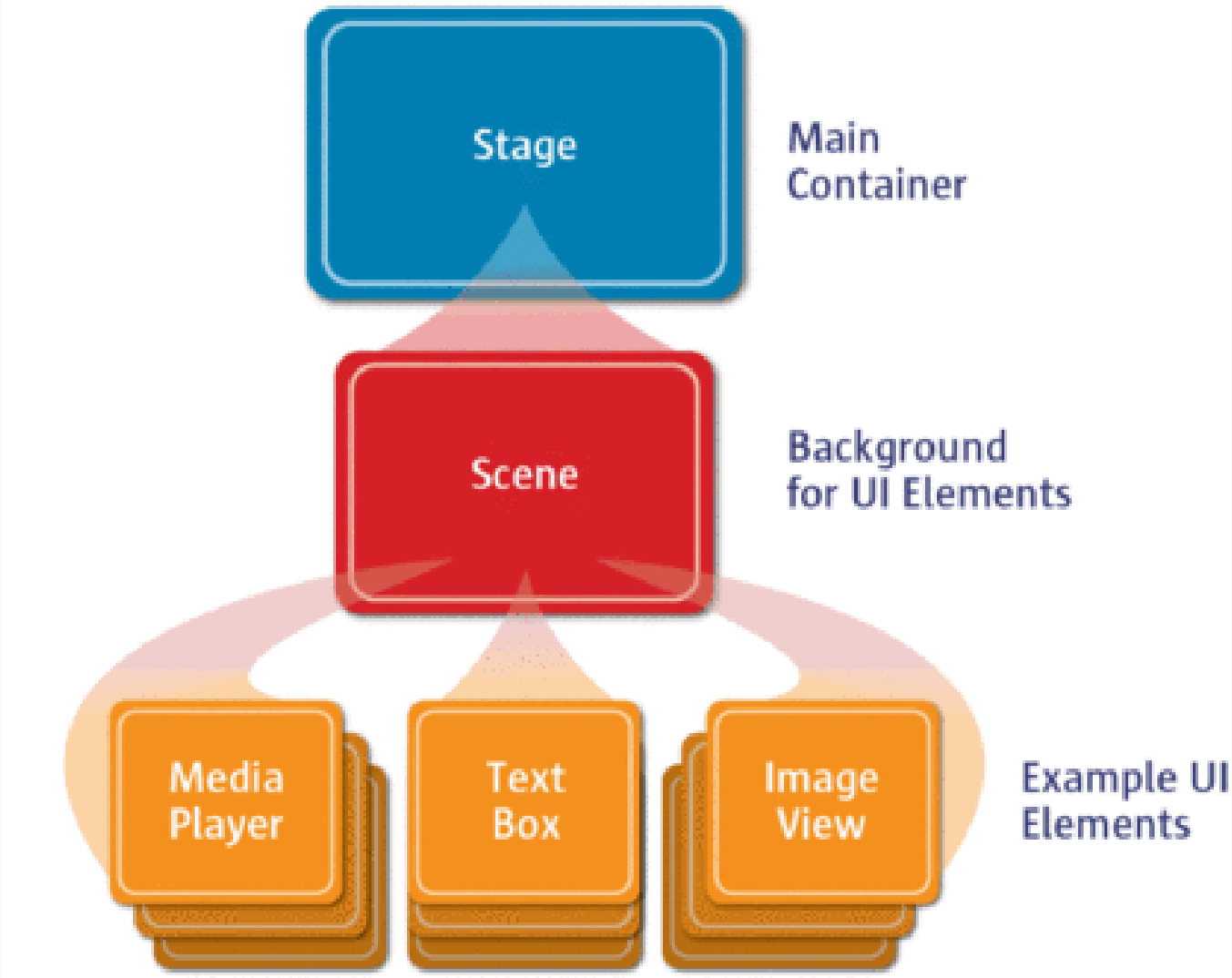
Pour éviter d'avoir une méthode start trop complexe, on peut définir une classe fille de Pane (ici de BorderPane) :

```
public class MyPane extends BorderPane {
    public MyPane() {
        Label lab = new Label("--> 0");
        Button button = new Button("OK");

        this.setCenter(lab);
        this.setBottom(button);
    }
}
```

Un code modulaire

```
public class PremiereGUI extends Application{
    @Override
    public void start(Stage stage)
                            throws Exception {
        stage.setTitle("GUI inutile");
        Pane myPane = new MyPane();

        Scene scene = new Scene(myPane);
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }

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

La hierarchie d'une application JavaFX



Source de l'image : www.oracle.com

Métaphore du théâtre : Stage

- Stage est la scène (au sens « physique » du terme), c'est-à-dire l'endroit où tout se passe : la fenêtre (délimitée par son cadre, et équipée de boutons pour la fermer, réduire, etc)
- Dans le cas d'une application « simple », avec une seule fenêtre, il n'y a qu'un seul objet Stage qui est créé
- Si le contenu de la fenêtre doit changer, ça veut dire qu'il faut remplacer la Scene

Métaphore du théâtre : Scene

- Scene est une scène de la pièce, c'est-à-dire un ensemble d'éléments de décor, installés à un endroit donné, à un moment donné, pour que les personnages (le/les utilisateur(s)) y agissent
- Une instance de Scene peut être vue comme une « configuration » possible de le fenêtre : si la fenêtre doit être modifiée, on remplace son objet Scene par un autre. Exemple :

```
Scene scene = new Scene(myPane);
stage.setScene(scene);
stage.sizeToScene();
stage.show();

// Ailleurs dans le code : nouvelle Scene
Scene scene2 = new Scene(otherPane);
stage.setScene(scene2);
stage.sizeToScene();
stage.show();
```

Métaphore du théâtre : le décor

- Le décor de la scène est constitué de l'ensemble de composants (boutons, labels, zones de texte, images,...)
- Pour simplifier la conception du décor, les éléments peuvent être groupés dans des panneaux
- Il y a différent types de panneaux, qui permettent différentes manières de disposer les composants
- la Scene a un panneau principal. Tous les composants peuvent être placés
  - soit directement dans le panneau principal de la Scene
  - soit dans des « sous-panneaux », qui sont situés dans le panneau principal

Control

- javafx.scene.control.Control
- Classe mère de l'ensemble des composants graphiques avec lequel l'utilisateur peut intéragir (cliquer, utiliser le clavier, déplacer la souris, scroller,...)
- On n'utilisera pas directement cette classe, mais plusieurs de ses descendantes

TextInputControl

- javafx.scene.control.TextInputControl
- Classe abstraite pour les zones d'entrée de texte
- Deux classes filles :
  - javafx.scene.control.TextArea est une zone dans laquelle l'utilisateur peut entrer du texte sur plusieurs lignes
  - javafx.scene.control.TextField est un champ de texte dans lequel l'utilisateur peut entrer du texte sur une ligne
- Remarque : dans les deux cas, c'est du texte non formaté. La classe javafx.scene.web.HTMLEditor permet la mise en forme du texte

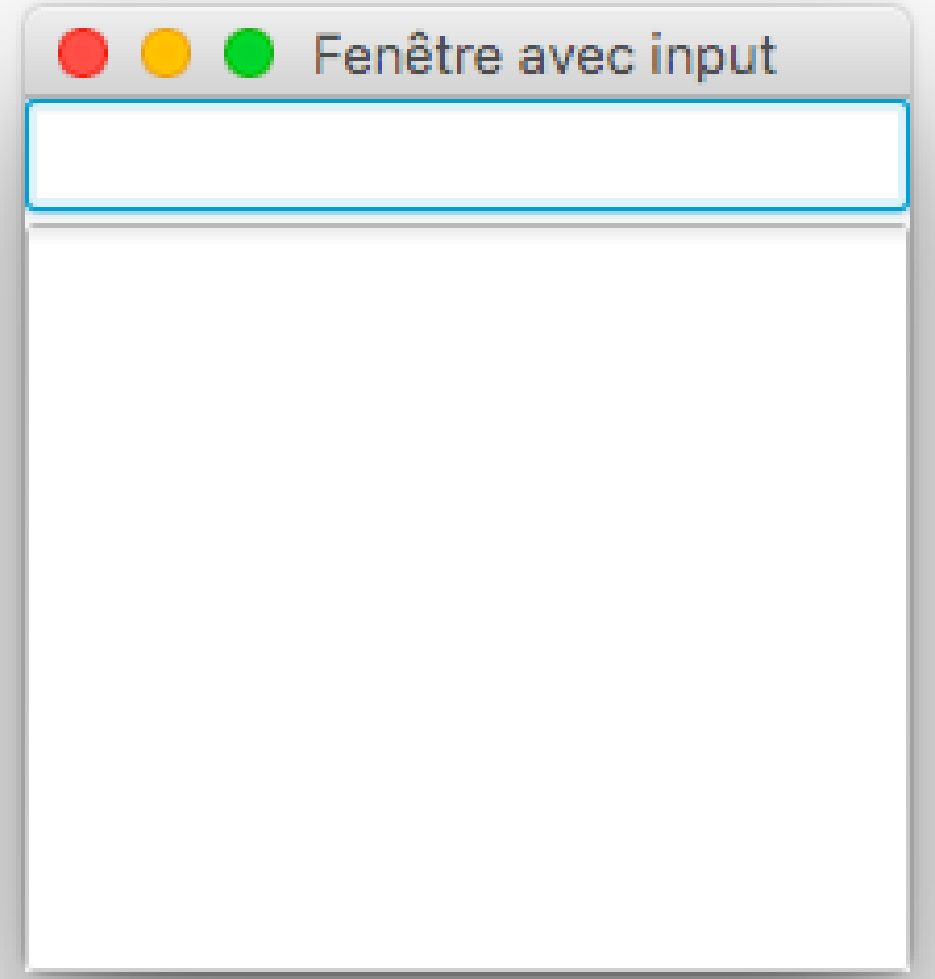
TextInputControl : Exemple

```
public class MyPaneWithInput extends BorderPane {
    public MyPaneWithInput() {
        TextField field = new TextField();
        TextArea area = new TextArea();
        area.setPrefRowCount(10);

        this.setTop(field);
        this.setBottom(area);
    }
}
```

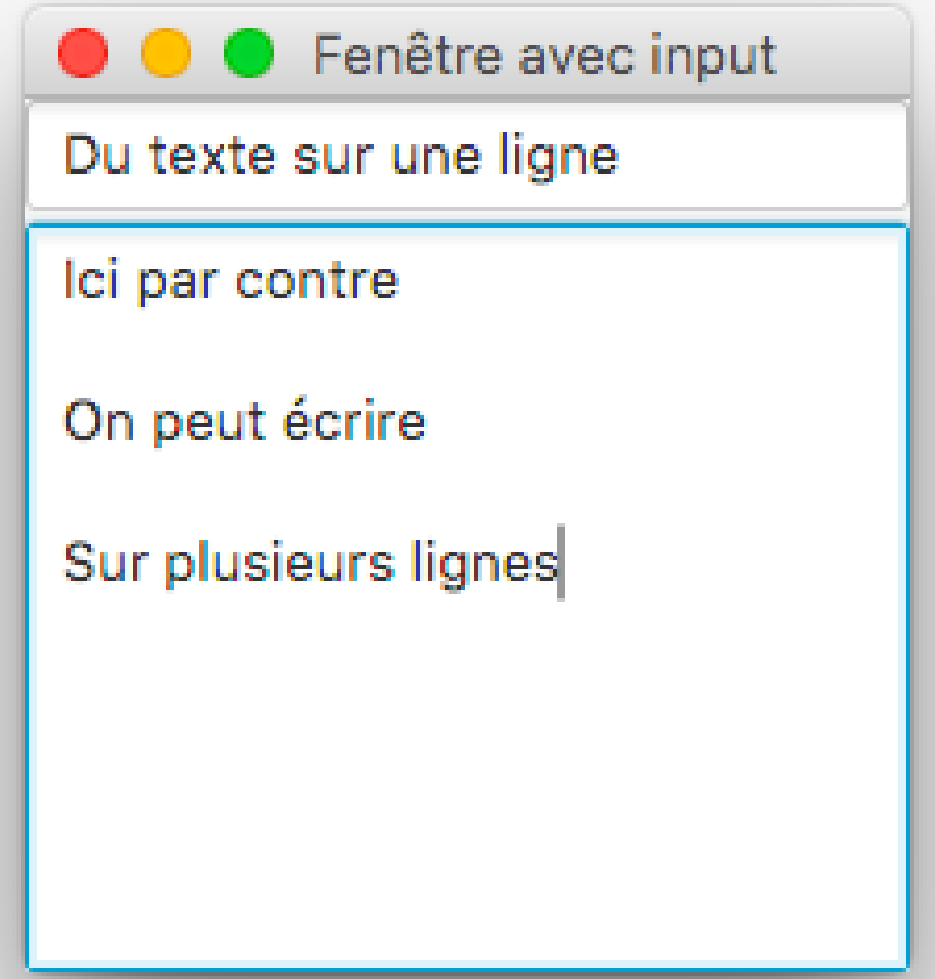
Et dans la méthode start :

```
Pane myPane = new MyPaneWithInput();
```

TextInputControl : Exemple

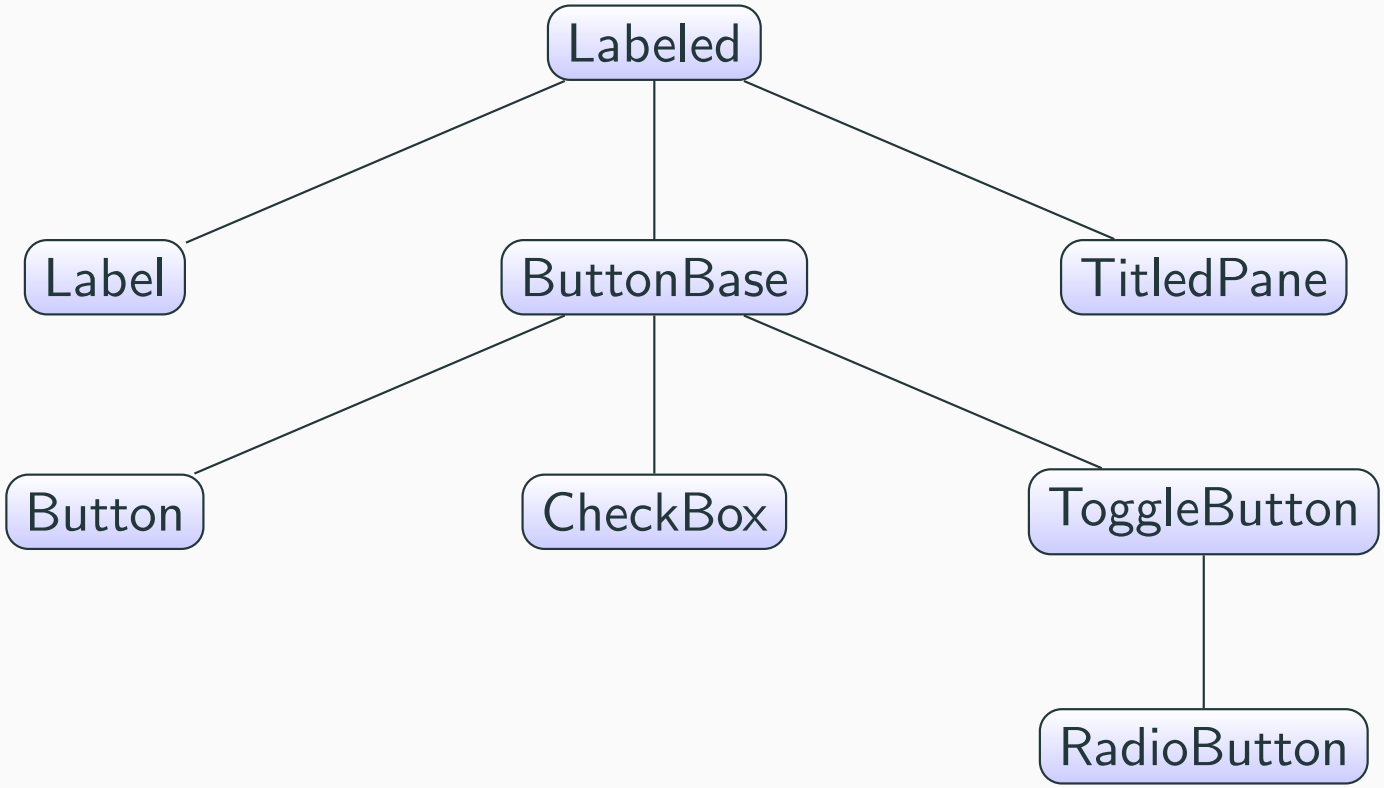


TextInputControl : Exemple



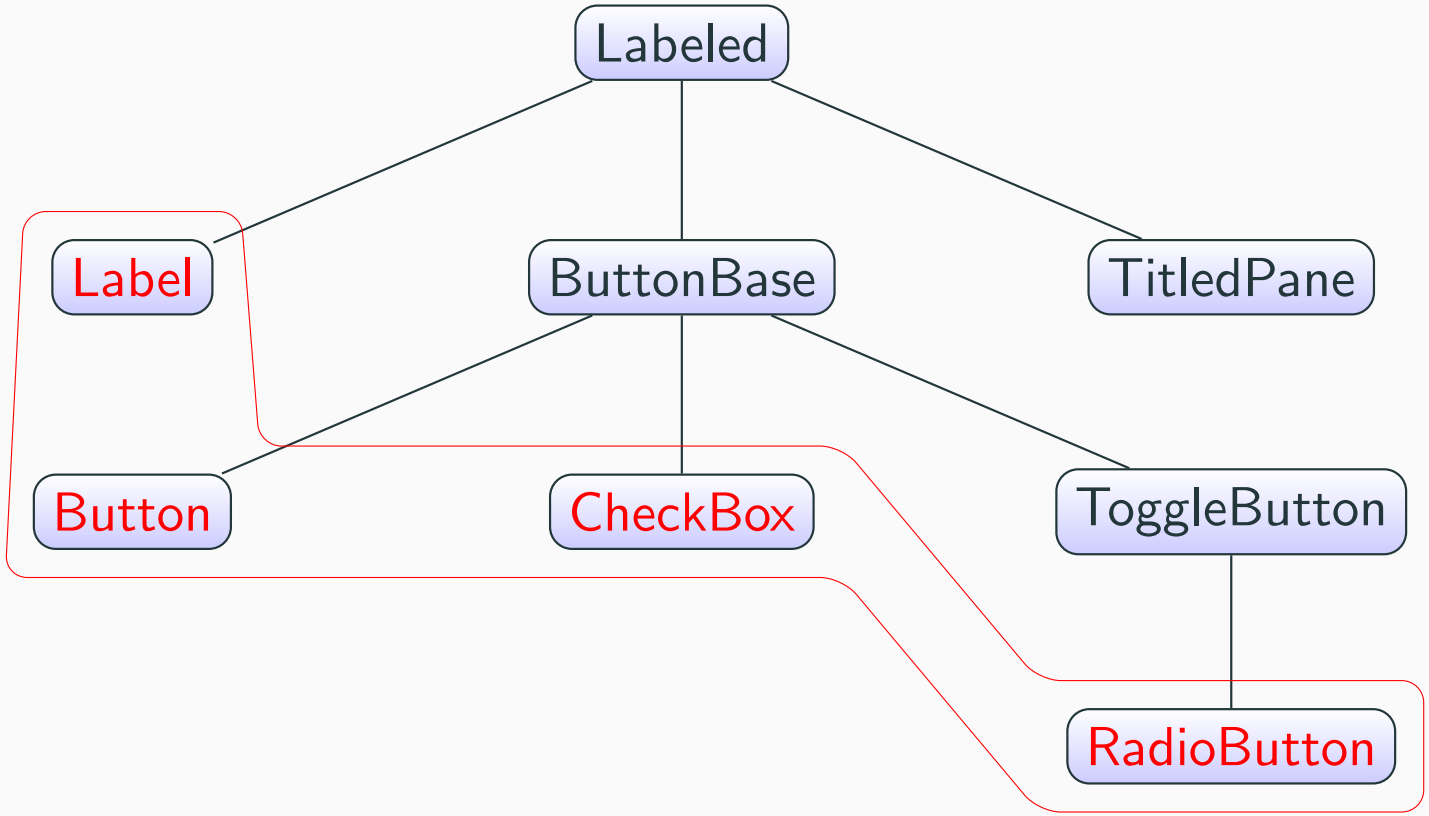
La hiérarchie de la classe Labeled

- javafx.scene.control.Labeled



La hiérarchie de la classe Labeled

- javafx.scene.control.Labeled



Label

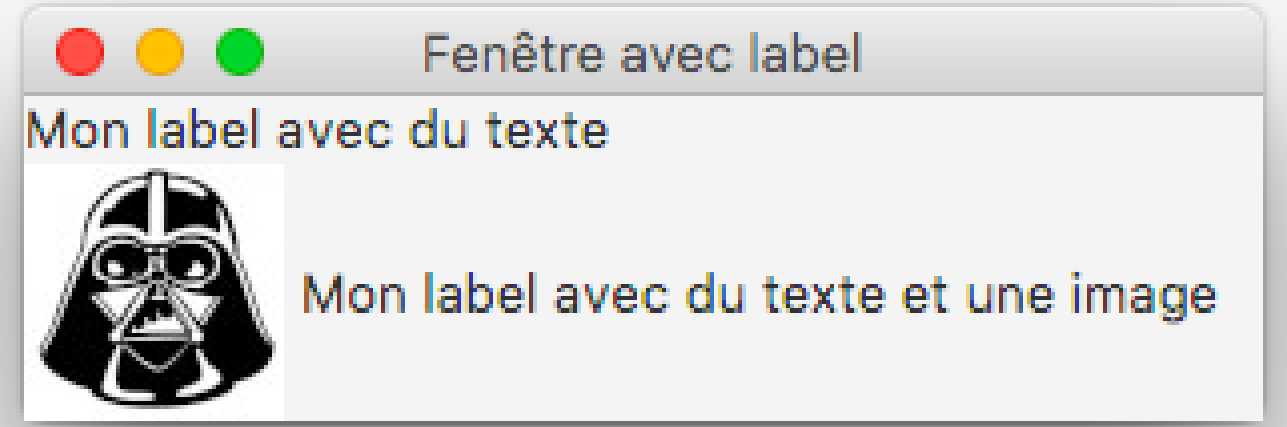
- javafx.scene.control.Label
- Zone de texte non éditable par l'utilisateur (mais peut-être modifié via le code Java)
- Constructeurs :
  - Sans paramètre : crée un label vide
  - Avec un String : crée un label avec un texte
  - Avec un String et un Node : crée un label avec un texte et une image

Label : exemple

```
public class MyPaneWithLabels extends BorderPane {
    public MyPaneWithLabels() {
        Label label1 = new Label("Mon label "
          + " avec du texte");
        // Initialisation de l'image
        // ...
        Label label2 = new Label("Mon label "
          + "avec du texte et une image",imageView);

        this.setTop(label1);
        this.setBottom(label2);
    }
}
```

- La méthode pour initialiser l'objet imageView sera vue plus tard

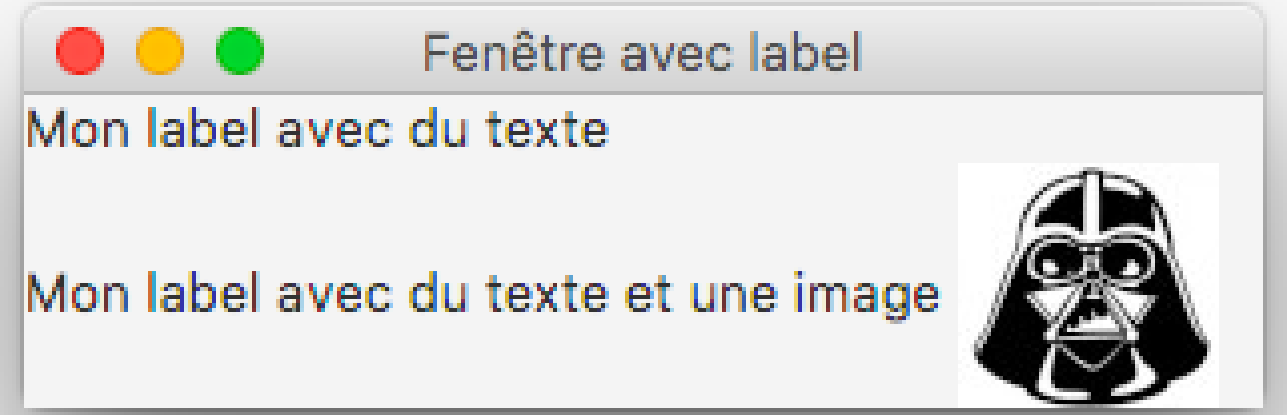
Label : exemple



- L'image est à gauche par défaut.
- On peut modifier ça avec la méthode setContentDisplay

Label : exemple

En ajoutant `label2.setContentDisplay(ContentDisplay.RIGHT);`



Modifier un Label

- setText(String s) remplace le texte actuel par s
- setGraphic(Node n) remplace l'image actuelle par n
- setContentDisplay(ContentDisplay c) modifie le positionnement de l'image par rapport au texte
- setTextFill (Paint p) modifie la couleur du texte
  - javafx.scene.paint.Paint est une classe de base pour représenter des couleurs et des dégradés
- ...