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

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) :

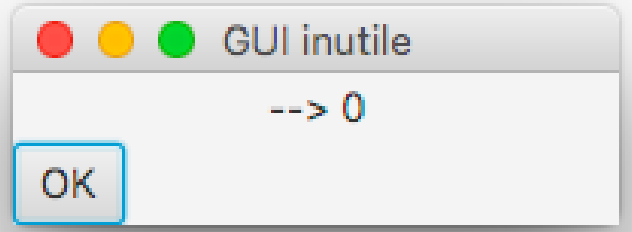
```
maMethode() {
   afficher("Hello World!")
}

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

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

Construire son interface

Premier exemple

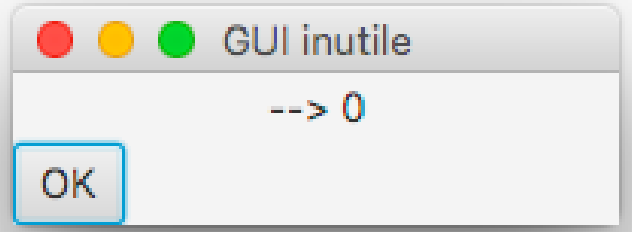
```
public class PremiereGUI extends Application {
    @Override
    public void start(Stage stage)
                                throws Exception {
        stage.setTitle("GUI inutile");
        BorderPane pane = new BorderPane();
        Label lab = new Label("--> 0");
        Button button = new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

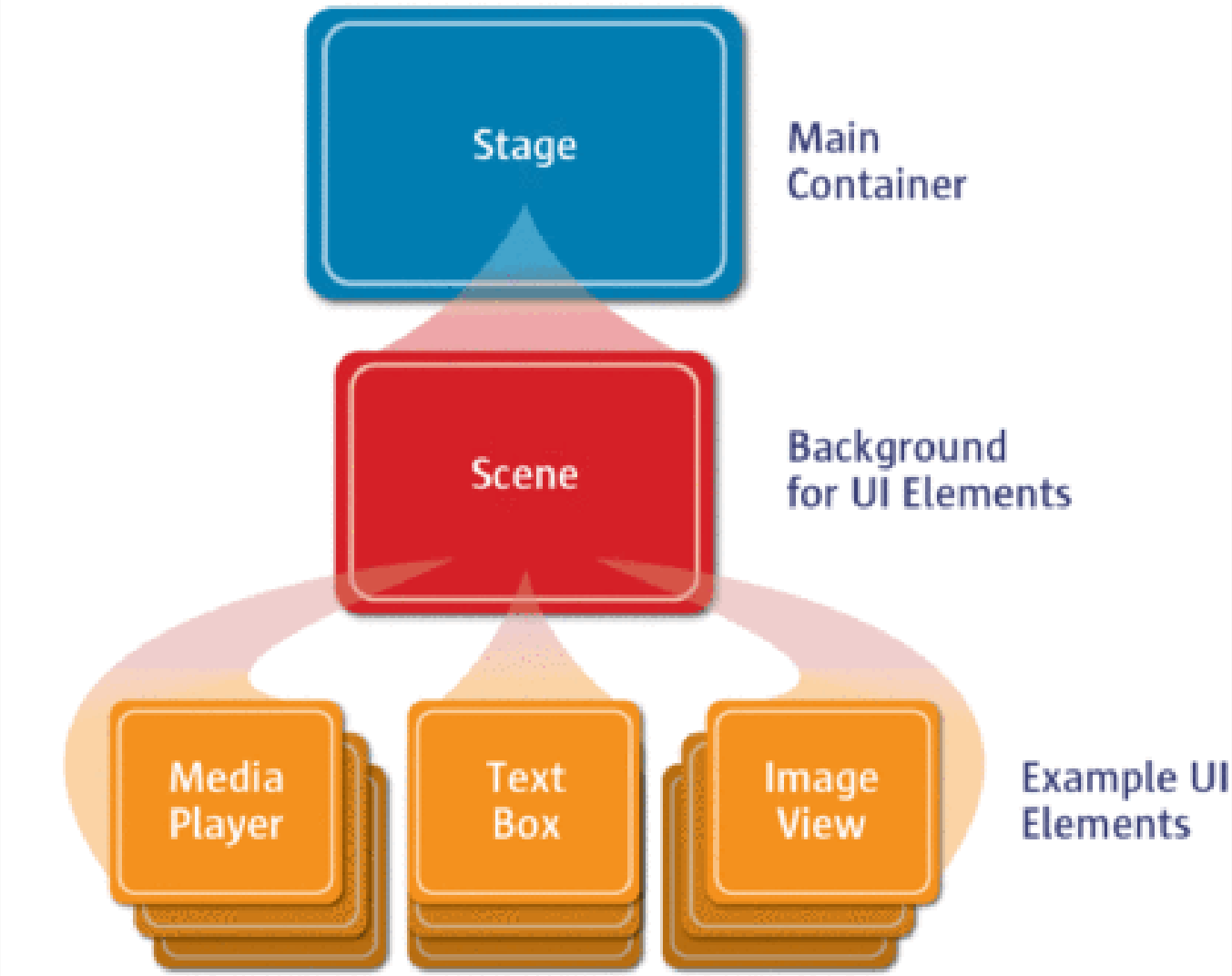
Pour éviter d'avoir une méthode start trop complexe, on peut définir une classe fille de Pane (ici de BorderPane) :

```
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

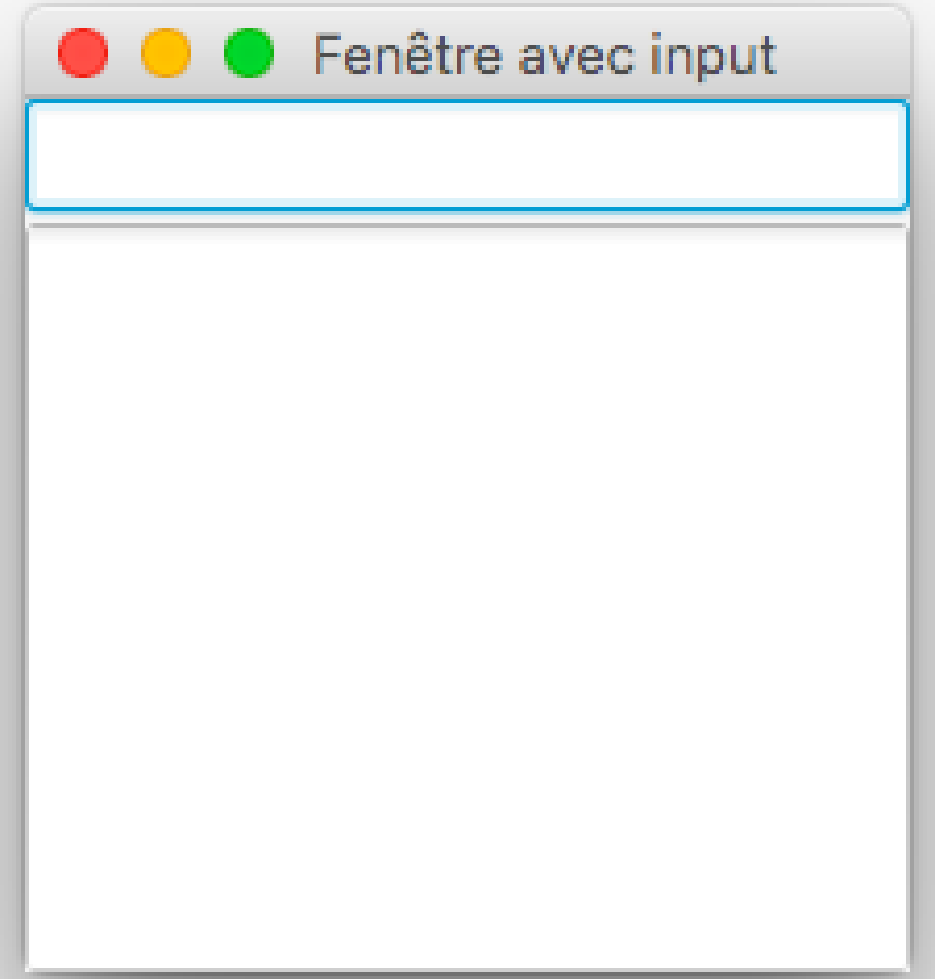
TextInputControl : Exemple

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

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

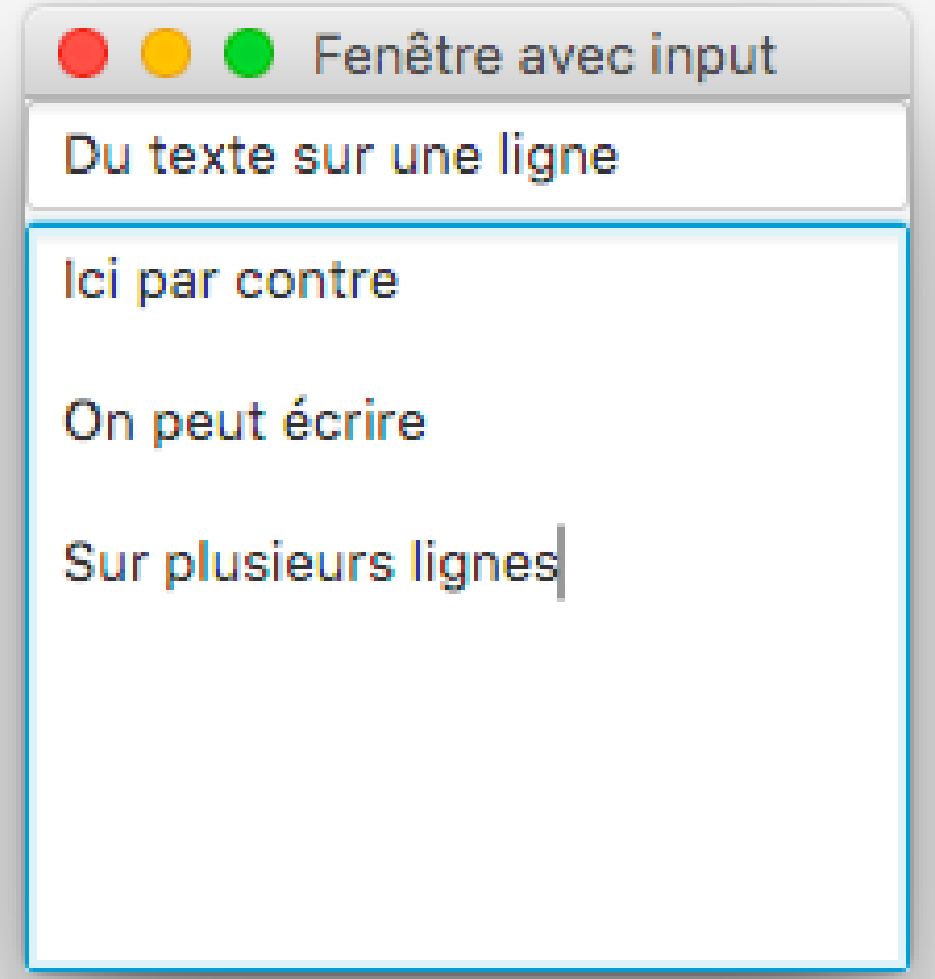
Et dans la méthode start :

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

TextInputControl : Exemple

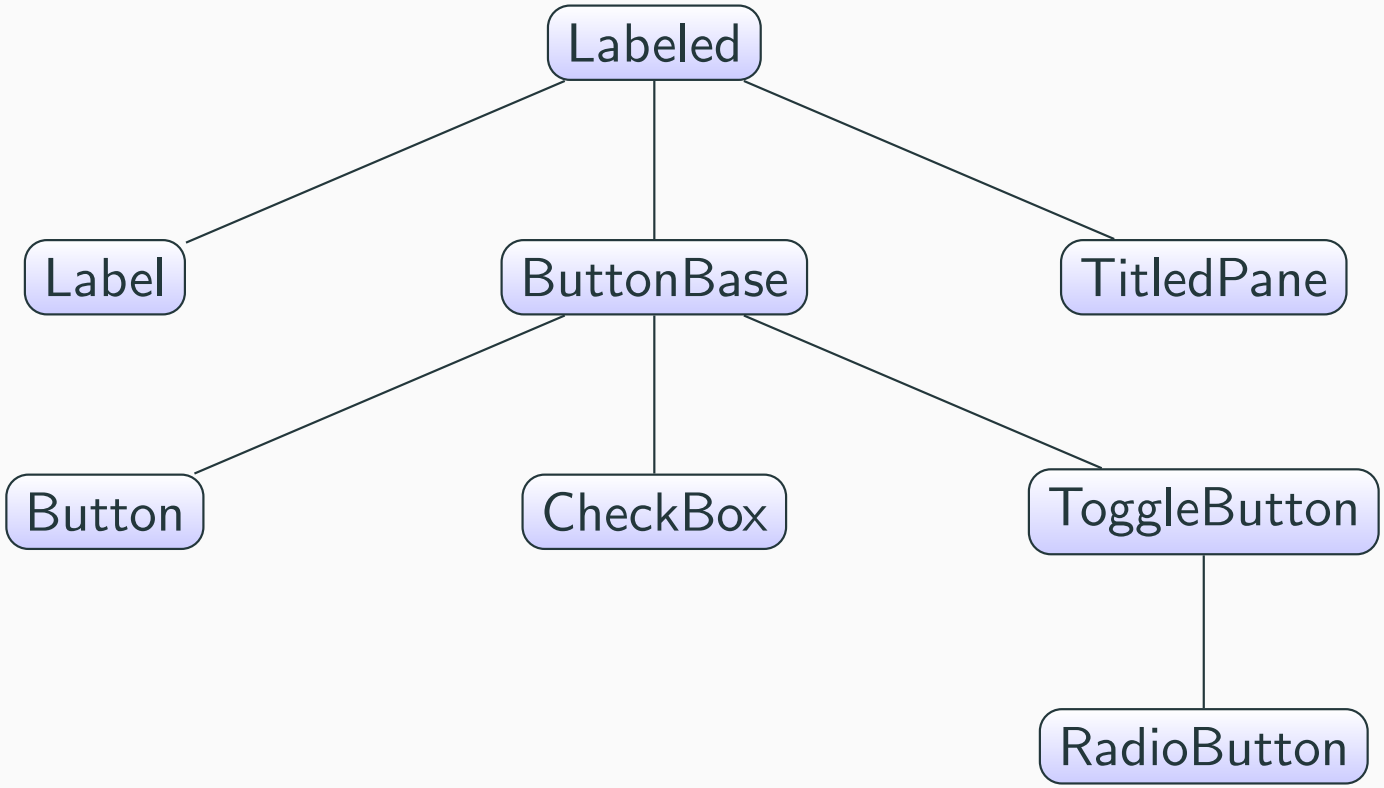


TextInputControl : Exemple



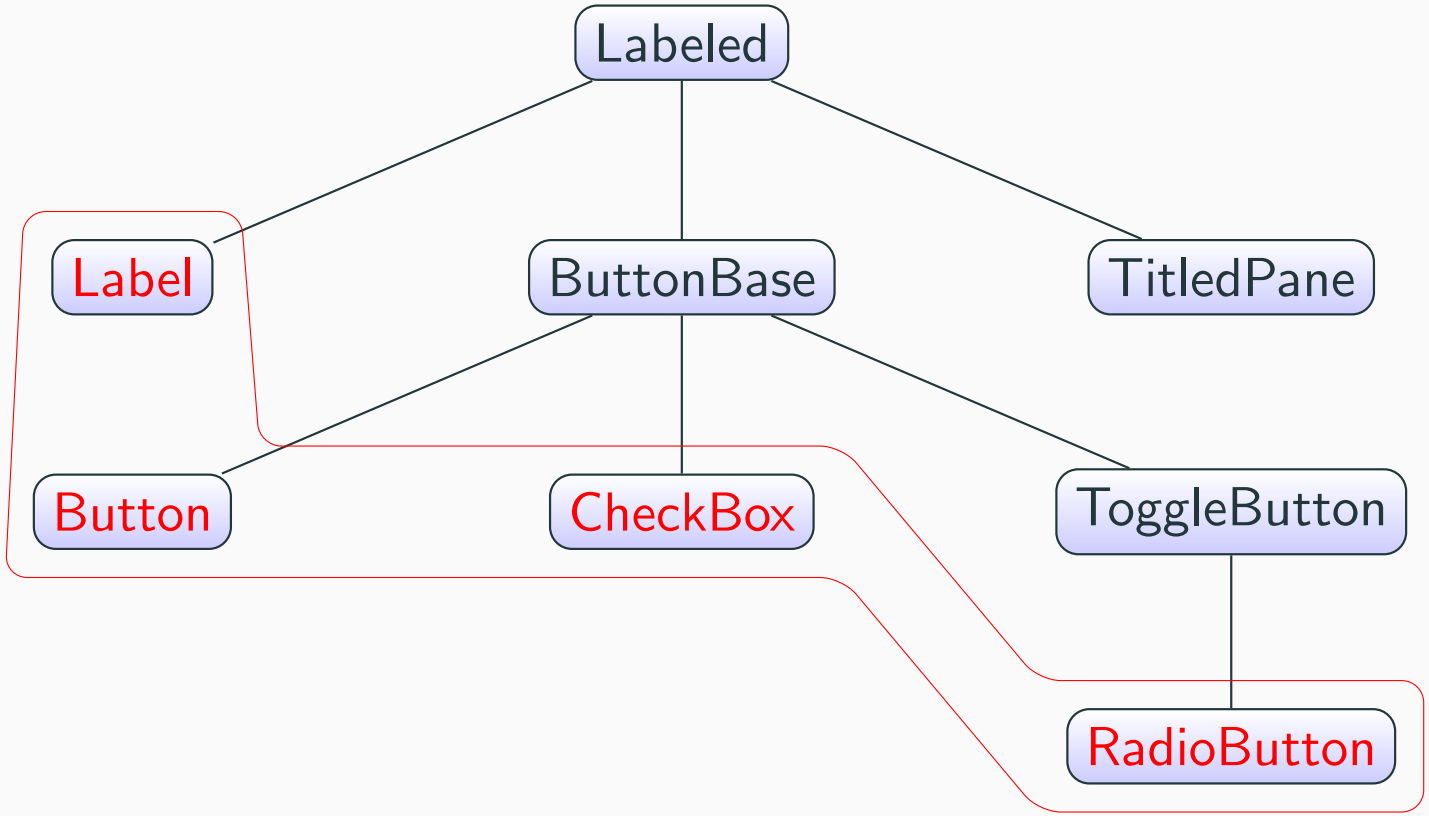
La hiérarchie de la classe Labeled

- javafx.scene.control.Labeled



La hiérarchie de la classe Labeled

- javafx.scene.control.Labeled



Label

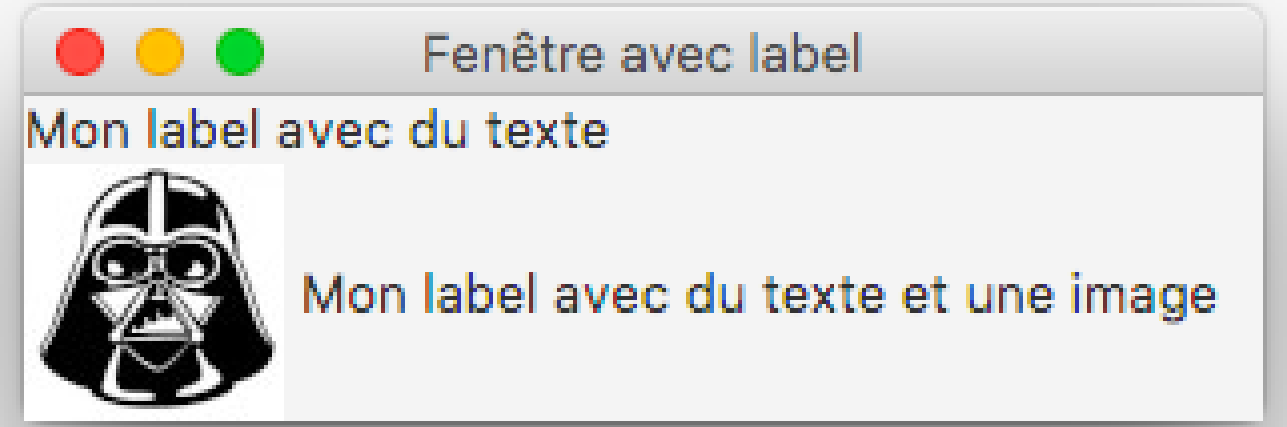
- javafx.scene.control.Label
- Zone de texte non éditable par l'utilisateur (mais peut-être modifié via le code Java)
- Constructeurs :
  - Sans paramètre : crée un label vide
  - Avec un String : crée un label avec un texte
  - Avec un String et un Node : crée un label avec un texte et une image

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

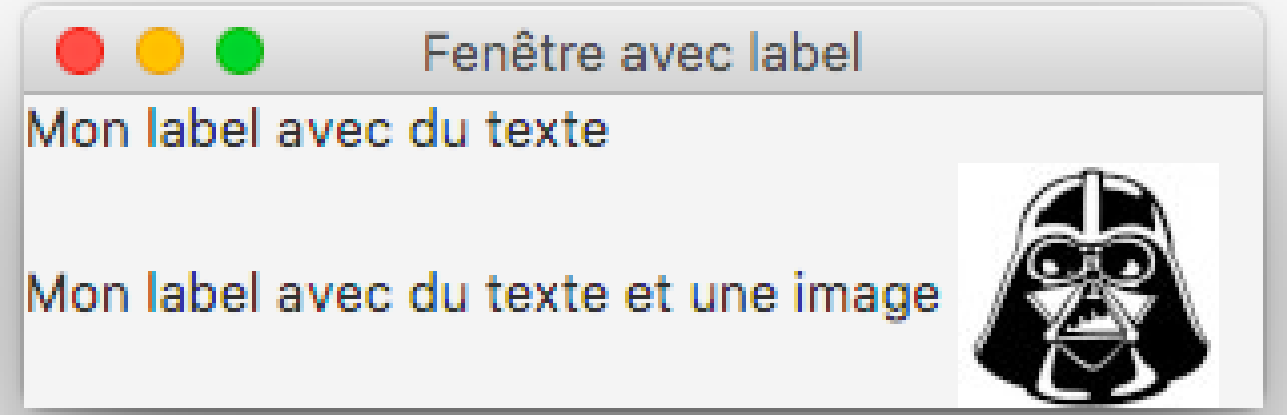
Label : exemple



- 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
- ...

Button

- javafx.scene.control.Button
- Bouton cliquable, peut contenir du texte et/ou une image
  - Constructeurs similaires au Label (sans paramètre, avec un String, ou avec un String et un Node)

Button : Exemple

```
public class MyPaneWithLabels extends BorderPane {
    public MyPaneWithLabels() {
        Button bouton1 = new Button("Texte");
        // Initialisation des images
        // ...
        Button bouton2 = new Button("Texte et image",
                                    yodaImageView);
        Button bouton3 = new Button("",vaderImageView);

        this.setTop(bouton1);
        this.setCenter(bouton2);
        this.setBottom(label3);
    }
}
```

Button : Exemple

Modifier un Button

Similaire au 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
- ...

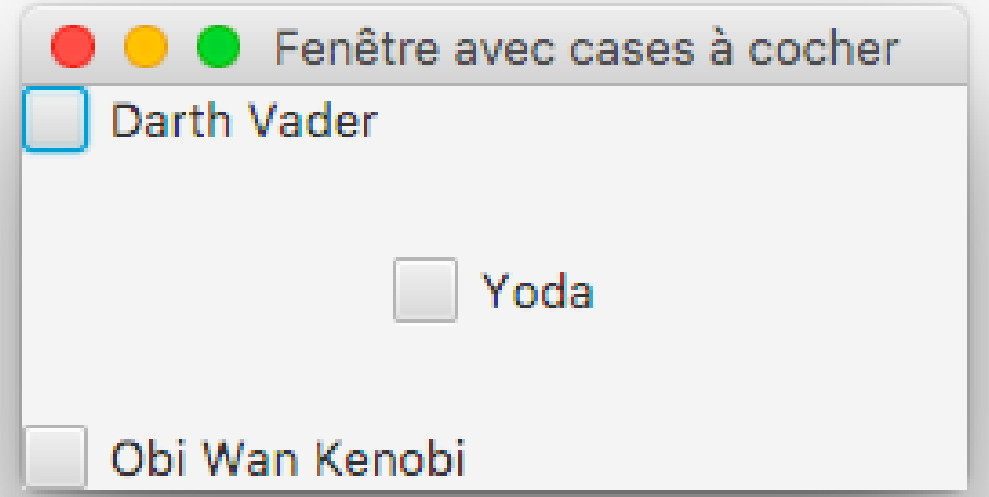
CheckBox

- javafx.scene.control.CheckBox
- Case à cocher par l'utilisateur
- On peut déterminer dans le programme si elle est cochée ou non
- Deux constructeurs :
  - Sans paramètre
  - Avec un String
  - Pas de constructeur avec String et Node cette fois !

CheckBox : Exemple

```
public class MyPaneWithCheckBox extends BorderPane {
    public MyPaneWithCheckBox() {
        CheckBox box1 = new CheckBox("Darth Vader");
        CheckBox box2 = new CheckBox("Yoda");
        CheckBox box3 = new CheckBox("Obi Wan Kenobi");

        this.setTop(box1);
        this.setCenter(box2);
        this.setBottom(box3);
    }
}
```

CheckBox : Exemple



CheckBox

En plus des méthodes habituelles pour la modifier :

- **boolean** isSelected() pour déterminer si la case est cochée
- **void** setSelected(**boolean** b) pour cocher/décocher la case

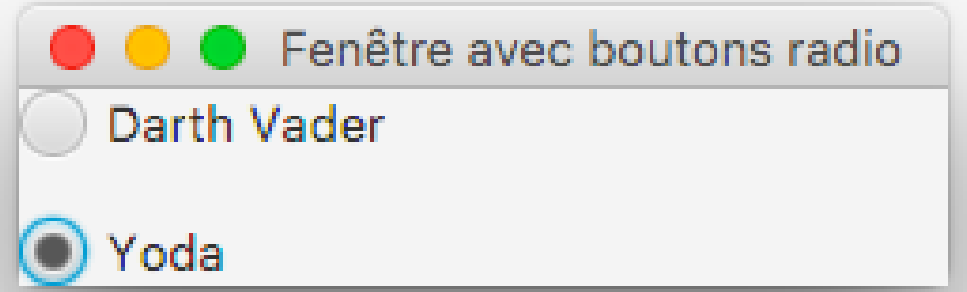
RadioButton

- javafx.scene.control.RadioButton
- Similaire à une case à cocher, sauf que
  - les boutons radio sont généralement placés en groupe
  - un seul bouton dans le groupe peut être sélectionné
- Deux constructeurs : sans paramètre, ou avec un String

RadioButton : Exemple

```
public class MyPaneWithRadioButton extends BorderPane {
    public MyPaneWithRadioButton() {
        ToggleGroup group = new ToggleGroup();
        RadioButton button1 = new RadioButton(
                                        "Darth Vader");
        button1.setToggleGroup(group);
        button1.setSelected(true);
        RadioButton button2 = new RadioButton("Yoda");
        button2.setToggleGroup(group);

        this.setTop(button1);
        this.setBottom(button2);
    }
}
```

RadioButton : Exemple



RadioButton : Méthodes

- Les habituelles méthodes de modification héritées de Labeled (setText, setTextFill ,...)
- setSelected et isSelected fonctionnent comme pour les cases à cocher
- **void** setToggleGroup(ToggleGroup value) permet d'indiquer à quel ToggleGroup le bouton radio appartient

Images

- javafx.scene.image.Image
- Charge une image à partir d'une URL. Exemples :
  - Image im = **new** Image("file:/chemin/vers/ fichier .png");
  - Image im = **new** Image("/fichier.png"); si le fichier est situé dans le classpath
  - Image im = **new** Image("http://www.unsite.fr/image.png"); pour une image disponible en ligne

Images

- javafx.scene.image.ImageView
- Permet d'afficher des images chargées grâce à la classe Image, mais aussi :
  - de modifier la taille de l'image
  - de conserver (ou modifier) le rapport longueur/largeur
  - restreindre l'affichage à une partie de l'image
  - ...

Images : Exemple

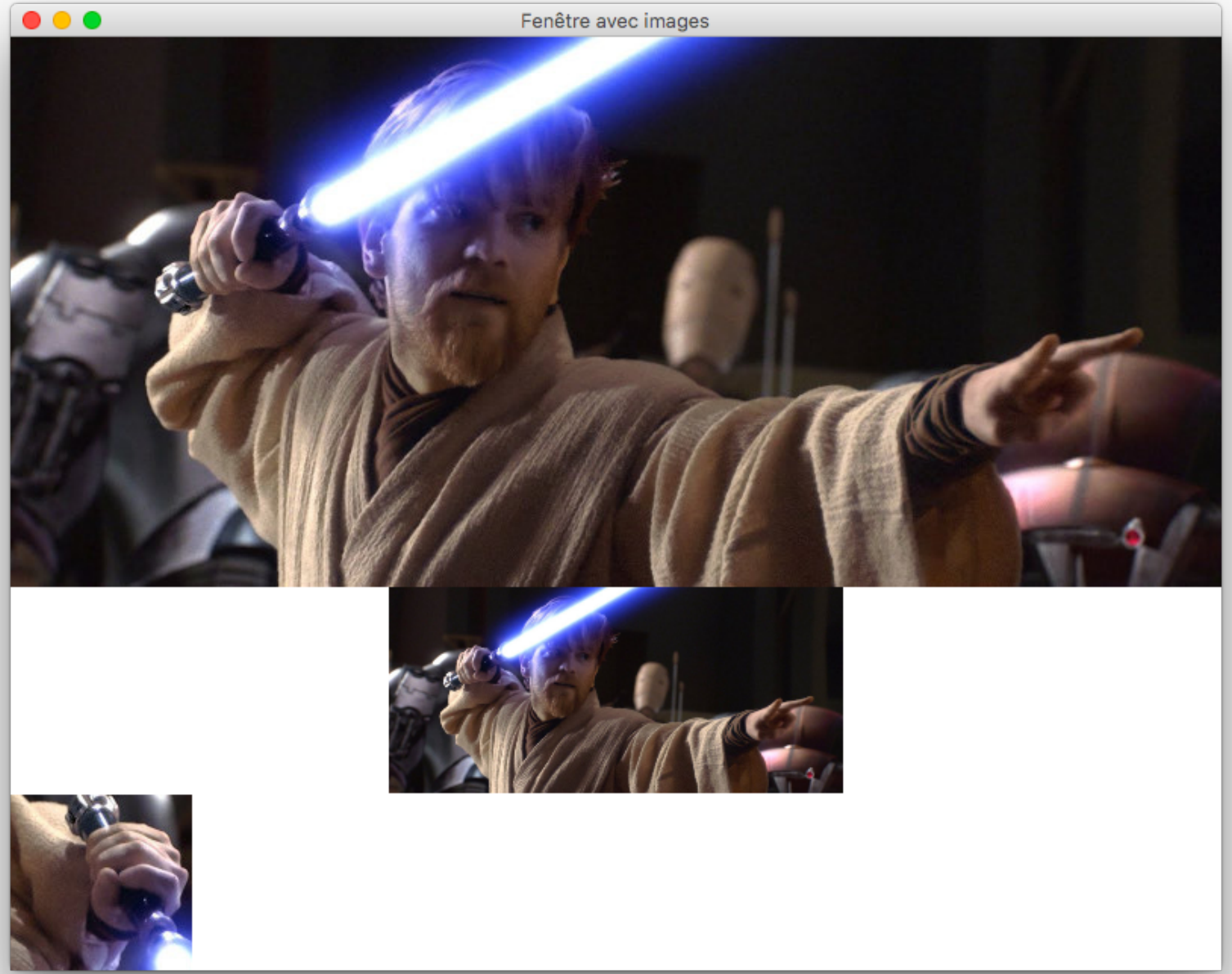
```
public class MyPaneWithImages extends BorderPane {
    public MyPaneWithImages() {
        Image image = new Image(
            "file:/chemin/vers/img_kenobi.jpg");

        ImageView iv1 = new ImageView();
        iv1.setImage(image);
        this.setTop(iv1);

        ImageView iv2 = new ImageView();
        iv2.setImage(image);
        iv2.setFitWidth(300);
        iv2.setPreserveRatio(true);
        this.setCenter(iv2);
```

Images : Exemple

```
        ImageView iv3 = new ImageView();
        iv3.setImage(image);
        Rectangle2D viewportRect = new Rectangle2D(
                                    100, 100, 120, 120);
        iv3.setViewport(viewportRect);
        iv3.setRotate(90);
        this.setBottom(iv3);
    }
}
```

Le Rectangle2D est utilisé pour sélectionner la zone de l'image à afficher

Images : Exemple



Pane et positionnement absolu

- javafx.scene.layout.Pane
- Classe de base pour regrouper des éléments (les enfants) dans un panneau
- Positionnement absolu

Pane et positionnement absolu : Exemple

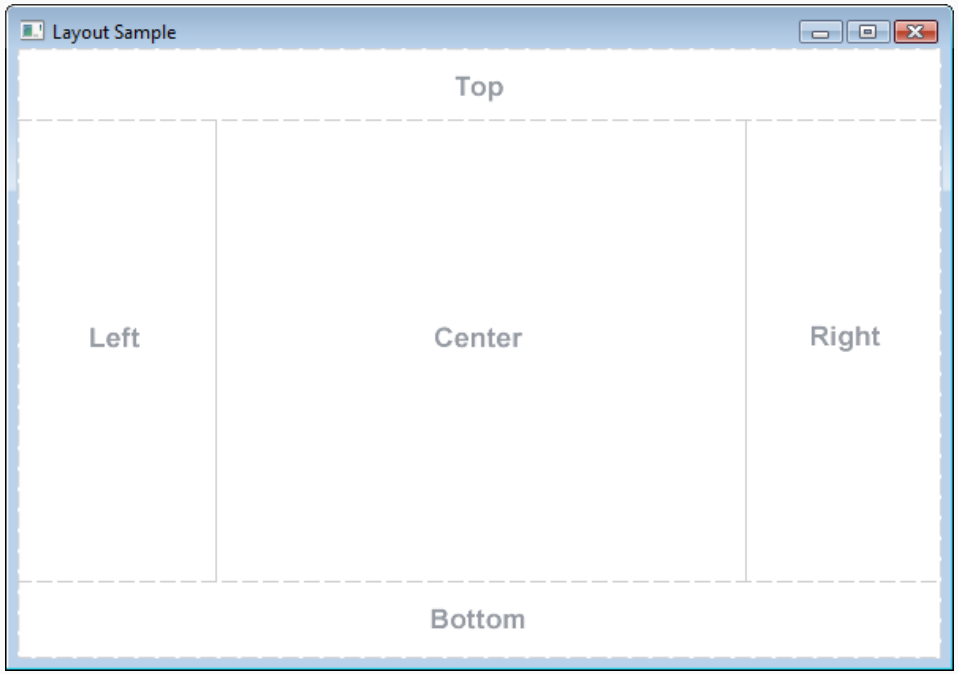
```
public class MyPaneWithAbsolutePosition extends Pane {
    public MyPaneWithAbsolutePosition() {
        this.setStyle("-fx-background-color: black;");
        this.setPrefSize(200, 200);
        Button b = new Button("bouton");
        b.relocate(20, 20);
        Label l = new Label("label");
        l.relocate(100, 100);
        this.getChildren().addAll(b,l);
    }
}
```

- setStyle peut modifier n'importe quel élément via CSS
- relocate permet d'indiquer les coordonnées de l'élément
- getChildren.add ou getChildren.addAll ajoutent les éléments au Pane

Pane et positionnement absolu : Exemple

BorderPane

- javafx.scene.layout.BorderPane
- Un panneau avec cinq enfants, qui peuvent être placés en haut, en bas, à gauche, à droite et au centre :



Source image : docs.oracle.com

- Méthodes setTop, setBottom, setLeft, setRight et setCenter

FlowPane

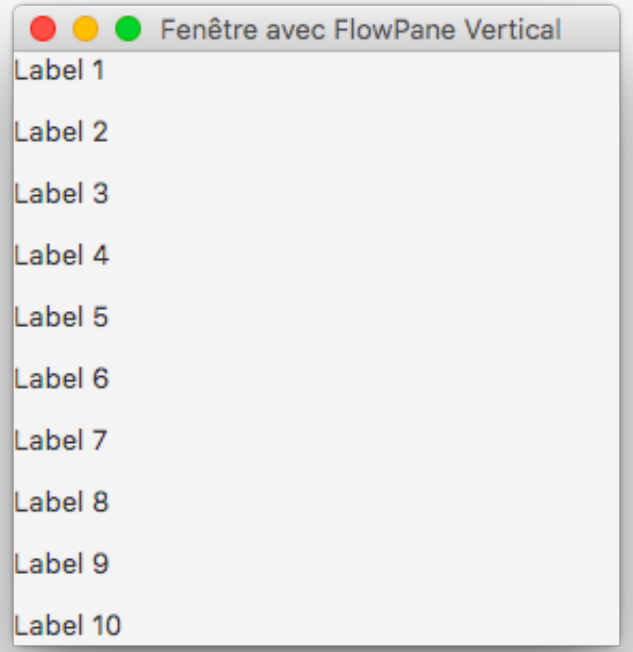
- javafx.scene.layout.FlowPane
- Place ses enfants successivement, en ligne ou en colonne
- Par défaut, un FlowPane est horizontal. Pour créer un FlowPane vertical, **new** FlowPane(Orientation.VERTICAL)
- Selon la taille de la fenêtre, les éléments peuvent être organisés en plusieurs lignes/colonnes
- setVgap et setHgap : pour fixer la distance entre les éléments

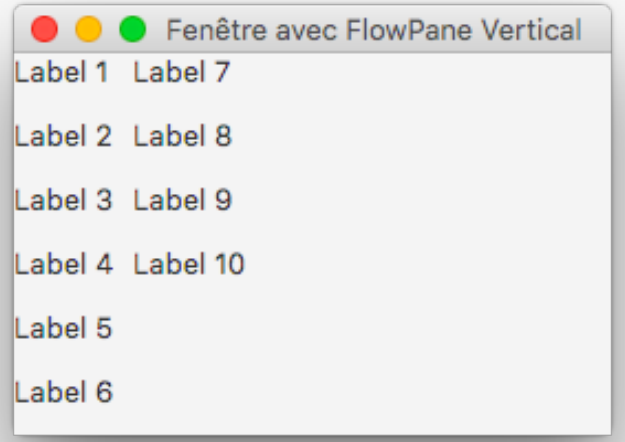
FlowPane : Exemple

```
public class MyFlowPane extends FlowPane {
    public MyFlowPane(boolean b) {
        if(b) setOrientation(Orientation.VERTICAL);
        setHgap(8);
        setVgap(12);

        for(int i = 1 ; i <= 10 ; i++) {
            Label lab = new Label("Label␣" + i);
            this.getChildren().add(lab);
        }
    }
}
```

FlowPane : Exemple de FlowPane vertical

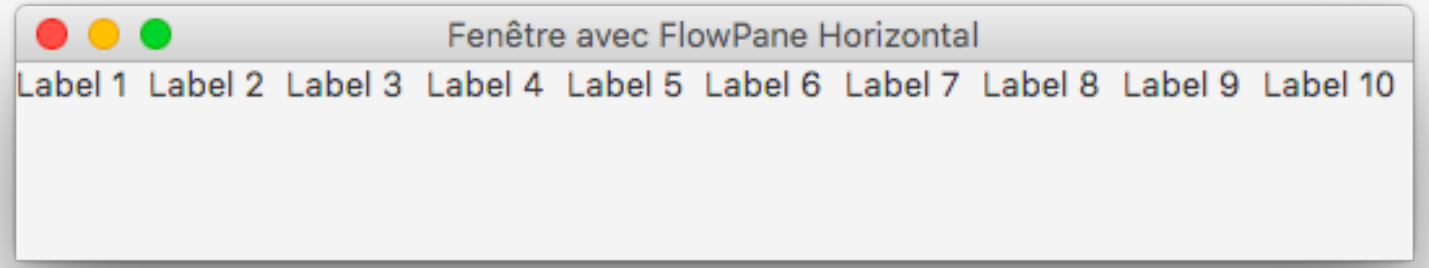
Pane myPane = **new** MyFlowPane(**true**);

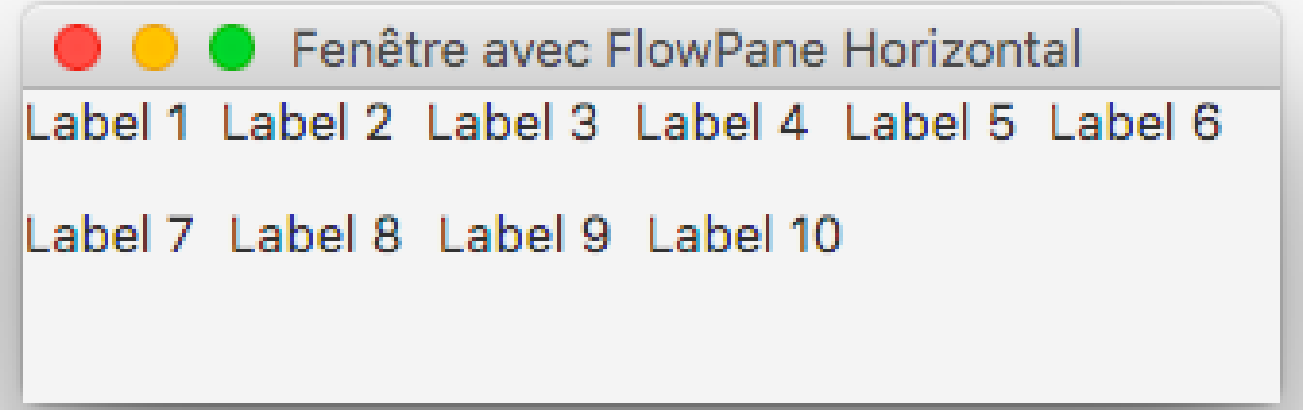




FlowPane : Exemple de FlowPane horizontal

Pane myPane = **new** MyFlowPane(**false**);





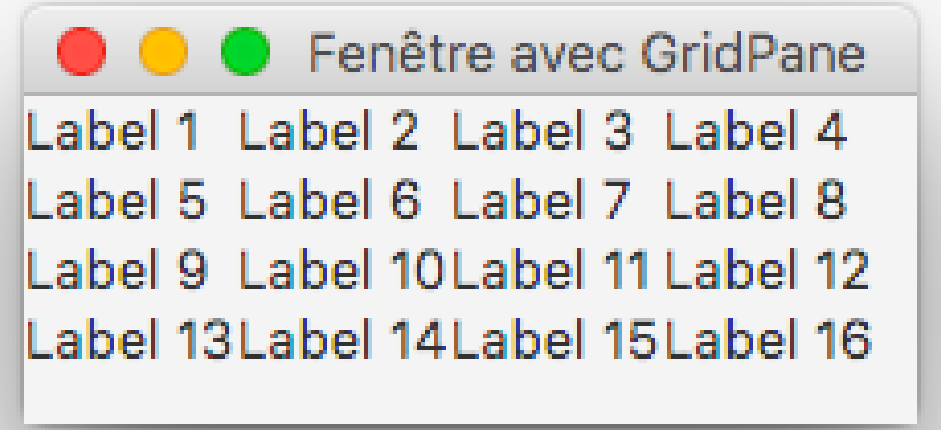
GridPane

- javafx.scene.layout.GridPane
- Placement des fils sous forme d'une grille
- Un élément fils peut être placé n'importe où dans la grille, et peut s'étaler sur plusieurs lignes et/ou colonnes
- Si l'indice de la ligne/colonne n'est pas spécifié, l'ajout d'un fils se fait dans la première case libre
- Un seul constructeur, sans paramètre

GridPane : Exemple

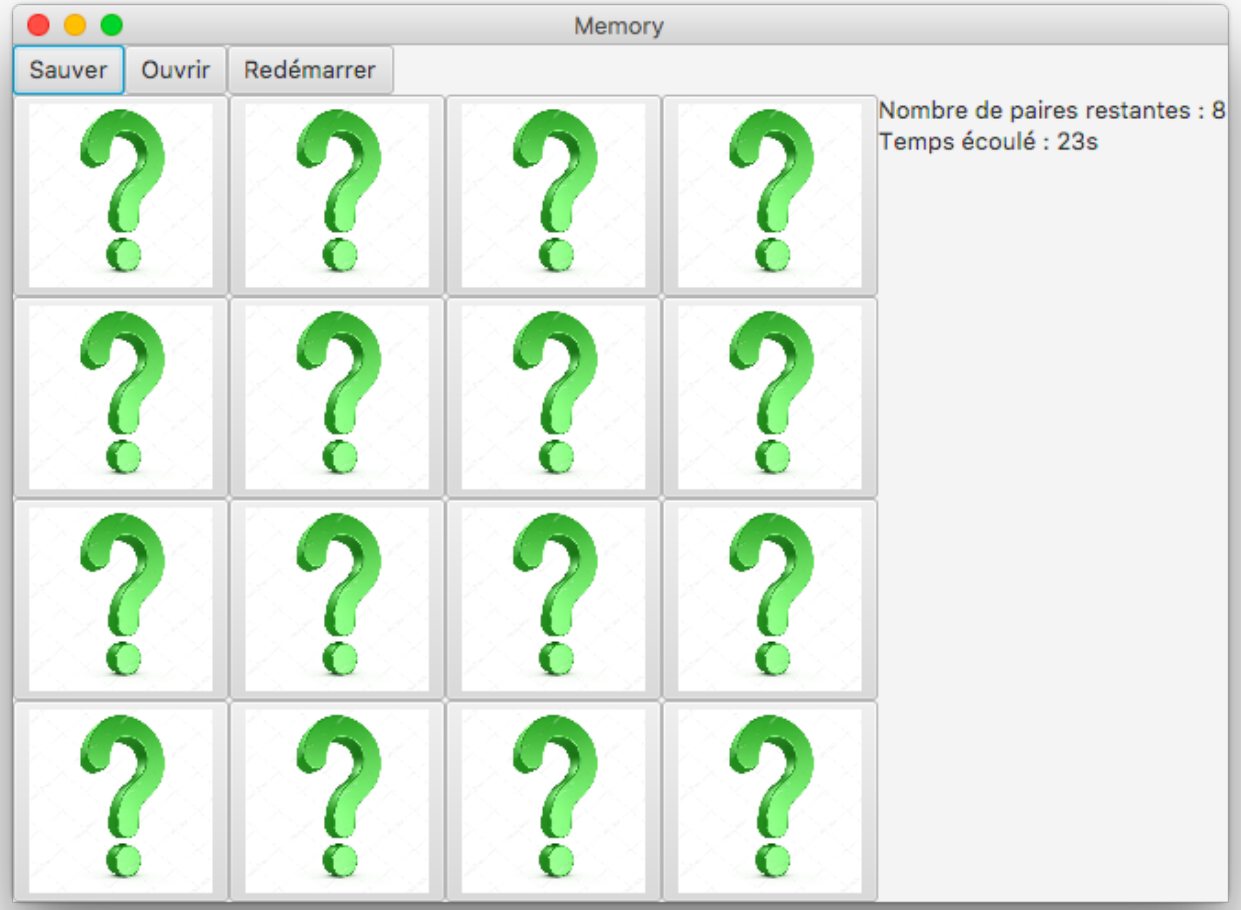
```
public class MyGridPane extends GridPane {
    public MyGridPane() {
        for(int i = 0 ; i < 16 ; i++) {
            Label lab = new Label("Label␣" + (i + 1));
            GridPane.setRowIndex(lab, i/4);
            GridPane.setColumnIndex(lab, i%4);
            this.getChildren().add(lab);
        }
    }
}
```

GridPane : Exemple



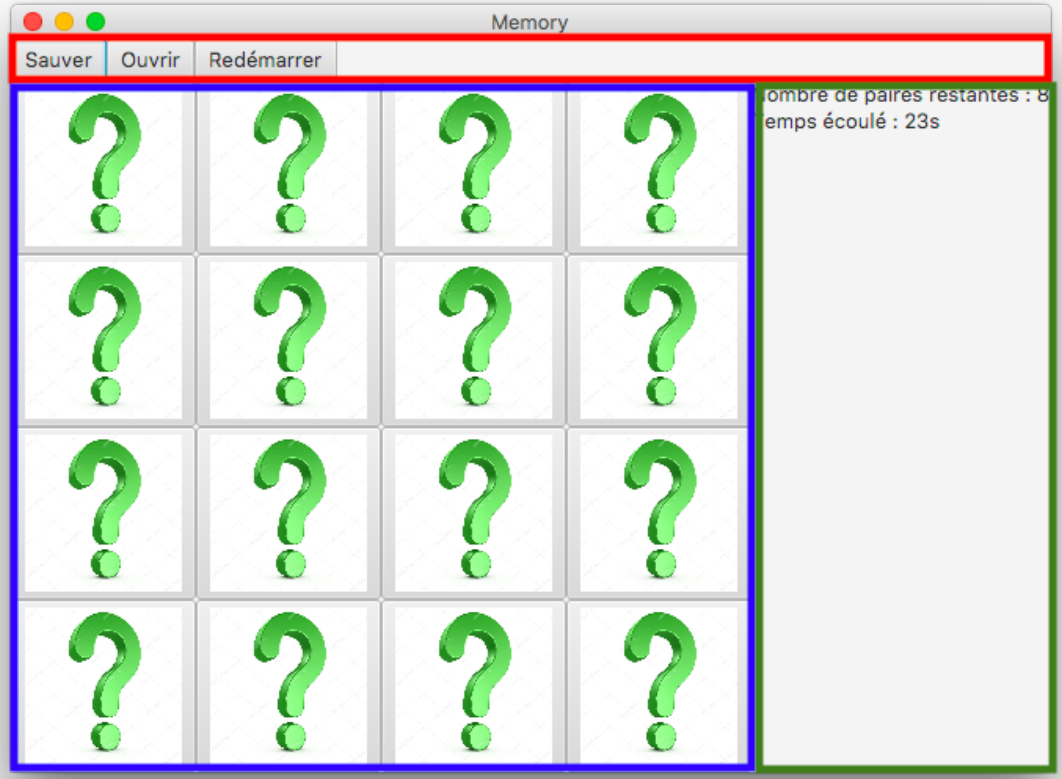
Combiner les panneaux

- Les fils d'un panneau peuvent être d'autres panneaux
- Cela permet d'agencer les éléments d'une manière complexe
- Exemple : Grille de Memory



Décomposition de la grille

Panneau principal : BorderPane



- top : Un FlowPane horizontal
- center : Un GridPane
- right : Un FlowPane vertical

Conclusion sur l'interface graphique

- Il existe d'autres éléments graphiques et d'autres Pane
- Voir la documentation officielle pour plus de détails (attention de bien prendre la version 11)

  https ://openjfx.io/javadoc/11/

- Exemples
  - HBox / VBox, tous les composants sont mis sur une ligne/une colonne (pas de passage à la ligne/colonne suivante)
  - Gestion des menus avec MenuBar, Menu, MenuItem, ContextMenu

Programmation événementielle

Événements et gestionnaires

- Événement : modification de l'état d'un élément
  - Clic sur un bouton
  - Déplacement de la souris
  - Sélection d'une case à cocher
  - Appui sur une touche du clavier
  - ...
- Gestionnaire (*handler*) : objet associé à un élément de la GUI, qui applique une méthode lorsqu'un événement survient
  - On parle aussi d'écouteur (*listener*) ou d'observateur (*observer*)

EventHandler

- EventHandler<T> est une interface paramétrée par un type T **extends** Event
- Lorsque l'événement a lieu, la méthode handle de EventHandler est appelée
- Différents types d'Event :
  - ActionEvent : clic sur un bouton, un item de menu, une case à cocher,... Associé à un élément par la méthode setOnAction
  - MouseEvent : événements de la souris. Méthodes pour associer le gestionnaire à un élément : setOnMouseClicked, setOnMouseMoved,...
  - KeyEvent : événements liés au clavier. Différentes méthodes pour différentes actions (setOnKeyTyped, setOnKeyPressed, setOnKeyReleased)

Clic sur un bouton : Compter les clics et modifier un label

```
public class CompteurClics extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("Compteur de Clics");
        Pane myPane = new CompteurPane();

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

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

Clic sur un bouton : Compter les clics et modifier un label

```
public class CompteurPane extends FlowPane {
    Label lab ;
    Button button ;
    Compteur compteur ;

    public CompteurPane() {
        lab = new Label("-->␣0");
        button = new Button("OK");
        compteur = new Compteur();

        button.setOnAction(new
            CompteurHandler(lab, compteur));

        this.getChildren().addAll(button,lab);
    }
}
```

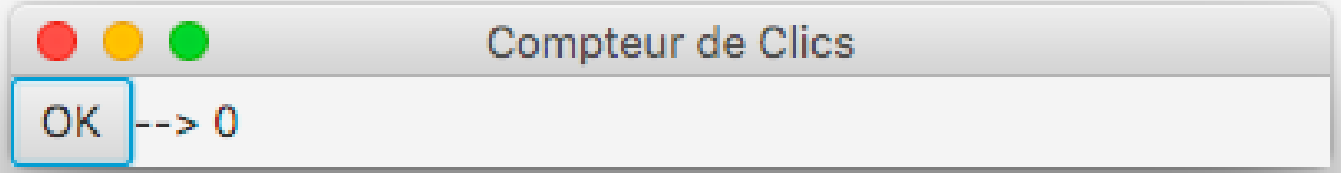
Clic sur un bouton : Compter les clics et modifier un label

```
public class Compteur {
    private int valeur ;

    public Compteur() { valeur = 0 ; }

    public void incrementer(int i) { valeur += i ; }

    public void incrementer() { valeur++ ; }

    public int getValeur() { return valeur ; }
}
```

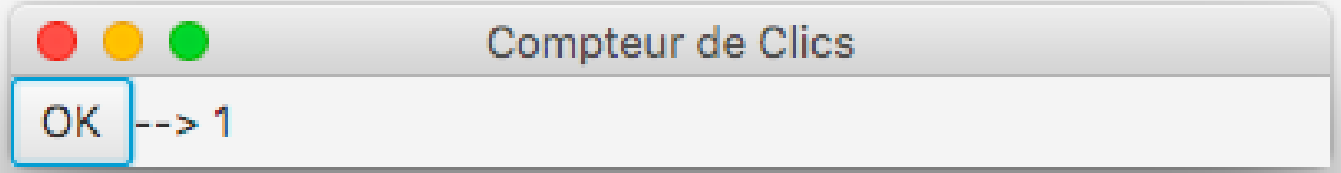
Clic sur un bouton : Compter les clics et modifier un label

```
public class CompteurHandler
    implements EventHandler<ActionEvent> {
  private Label lab ;
  private Compteur compteur ;

  public CompteurHandler(Label lab, Compteur cpt) {
    this.lab = lab ;
    compteur = cpt;
  }

  @Override
  public void handle(ActionEvent event) {
    compteur.incrementer();
    lab.setText("--> " + compteur.getValeur());
  }
}
```

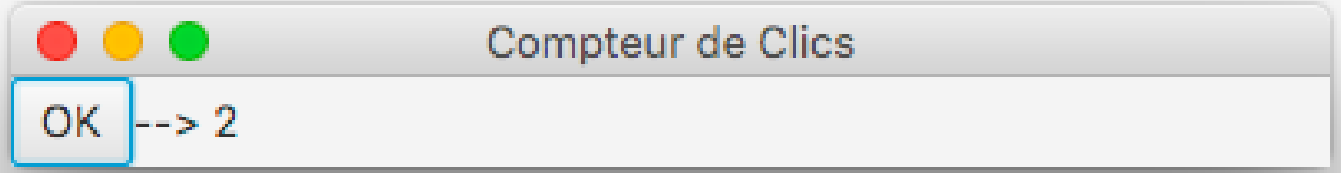
Clic sur un bouton : Compter les clics et modifier un label



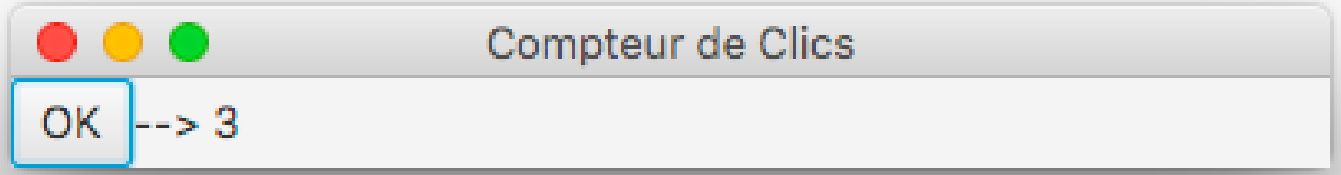
Clic sur un bouton : Compter les clics et modifier un label



Clic sur un bouton : Compter les clics et modifier un label



Clic sur un bouton : Compter les clics et modifier un label



Utiliser l'état d'une CheckBox

On modifie CompteurPane :

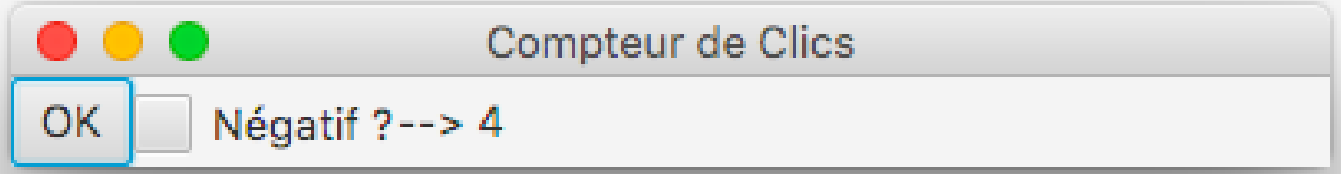
```
public class CompteurPane extends FlowPane {
    // ...
    CheckBox box ;

    public CompteurPane() {
        // ...
        box = new CheckBox("Negatif ?");
        button.setOnAction(new CompteurHandler(lab,
                box, compteur));
        this.getChildren().addAll(button, box,lab);
    }
}
```

Utiliser l'état d'une CheckBox

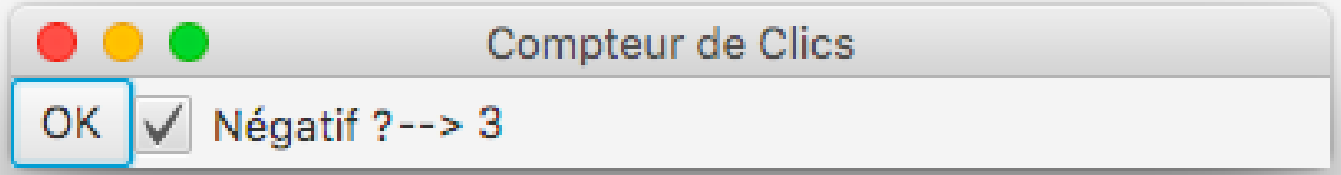
On modifie CompteurHandler :

```
public class CompteurHandler
        implements EventHandler<ActionEvent> {
    \\ ...
    private CheckBox box ;
    public CompteurHandler(Label lab, CheckBox box,
                                    Compteur compteur) {
        this.box = box ;
    }
    @Override
    public void handle(ActionEvent event) {
        if(box.isSelected()) compteur.incrementer(-1);
        else compteur.incrementer();
        lab.setText("--> " + compteur.getValeur());
    }
}
```

Utiliser l'état d'une CheckBox



Utiliser l'état d'une CheckBox



Utiliser le contenu d'un TextField

On modifie CompteurPane :

```
public class CompteurPane extends FlowPane {
    // ...
    TextField field ;

    public CompteurPane() {
        // ...
        field = new TextField();
        button.setOnAction(new CompteurHandler(
                        lab, box, field, compteur));
        this.getChildren().addAll(button, box,
                                        field,lab);
    }
}
```

Utiliser le contenu d'un TextField

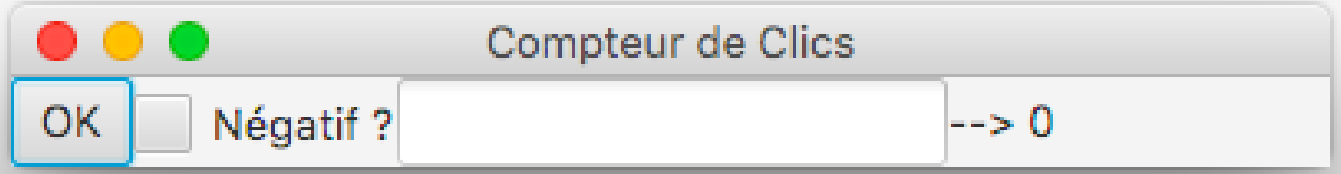
On modifie CompteurHandler :

```
public class CompteurHandler implements
                    EventHandler<ActionEvent> {
    // ...
    private TextField field;

    public CompteurHandler(Label lab, CheckBox box,
                        TextField field, Compteur compteur) {
        // ...
        this.field = field;
    }
```

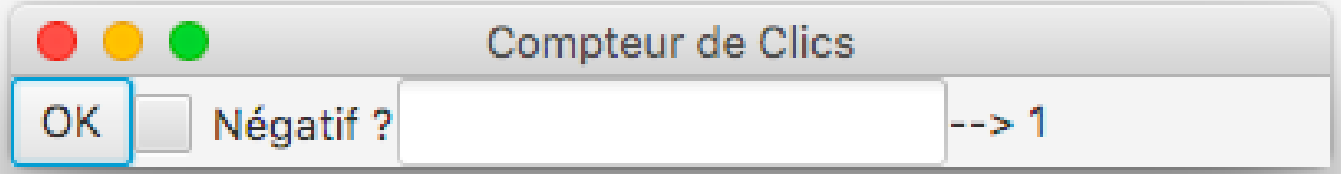
Utiliser le contenu d'un TextField

```
    @Override
    public void handle(ActionEvent event) {
        int val = 1;
        if (! "".equals(field.getText()))
            val = Integer.parseInt(field.getText());
        if (box.isSelected())
            compteur.incrementer(-1 * val);
        else
            compteur.incrementer(val);
        lab.setText("-->␣" + compteur.getValeur());
    }
}
```

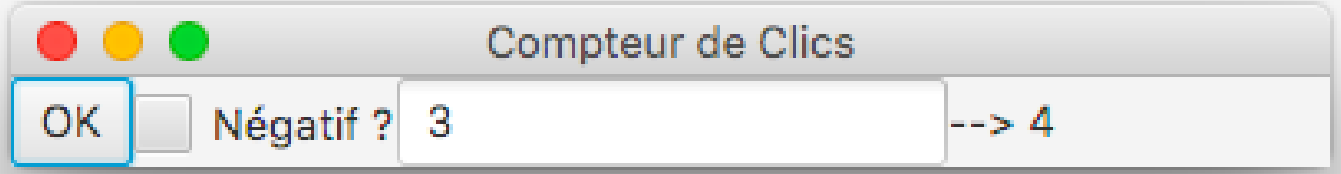
Utiliser le contenu d'un TextField



Utiliser le contenu d'un TextField



Utiliser le contenu d'un TextField



Utiliser le contenu d'un TextField