

Développement d'application avec IHM

Les composants graphiques



Christine Jacquin

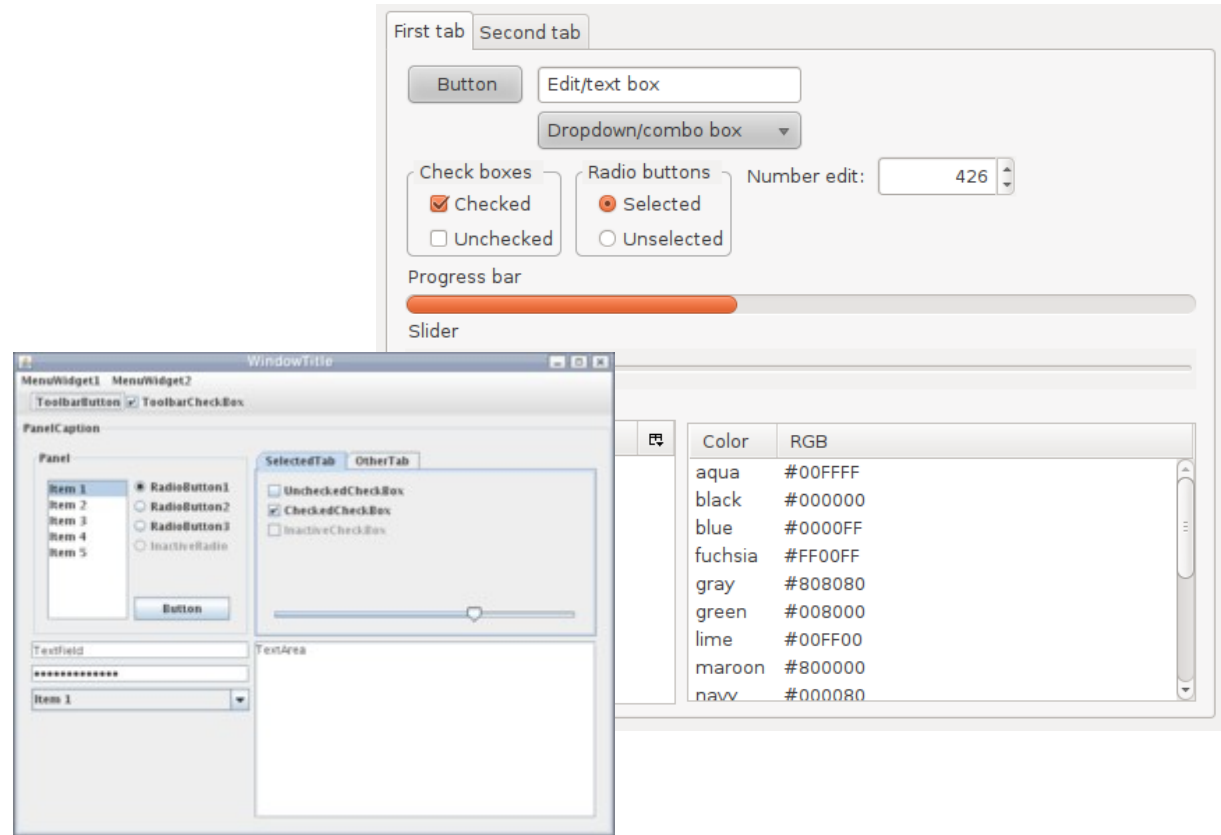
Développement d'applications avec IHM

Intervenants en TD :

- Jean-François Berdjugin
- Jean-François Remm

Responsable du cours :

- Christine Jacquin

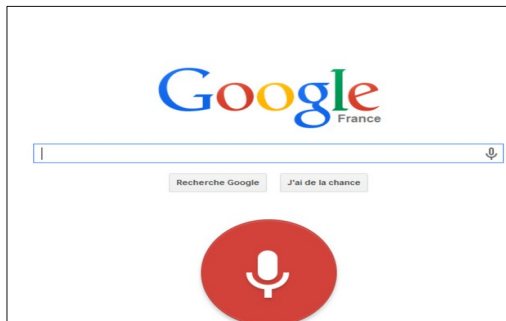


Interface homme machine

- Un ensemble de dispositifs matériels et logiciels permettant au couple (machine, utilisateur) de communiquer en vue de l'accomplissement d'une tâche de l'utilisateur
- Ergonomie
 - Accessibilité

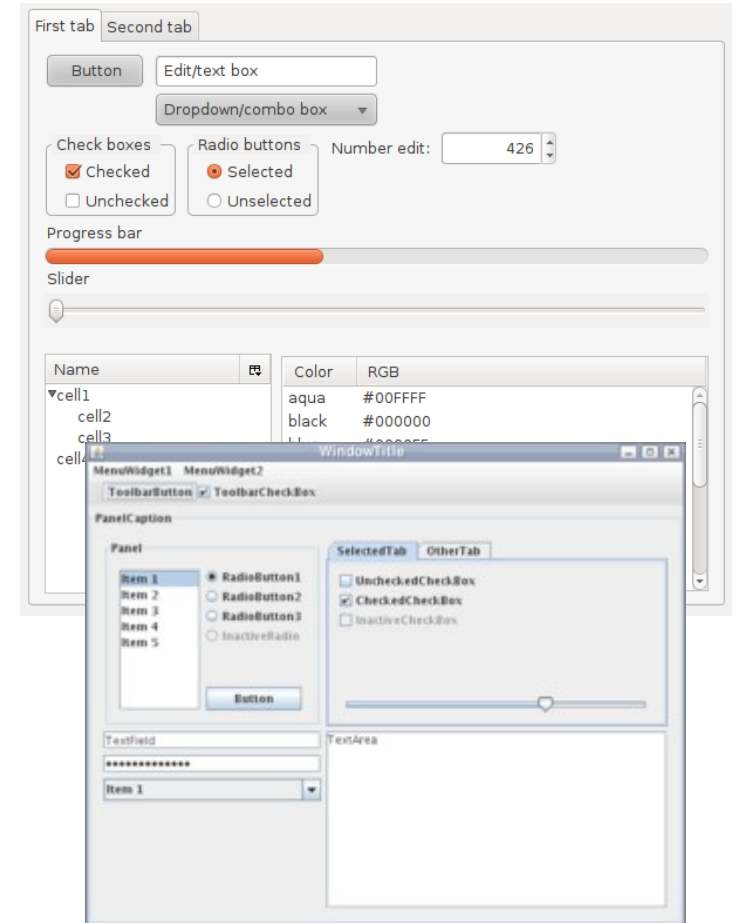
Interactions avec la machine

- clavier, souris, manette de jeux, écran, tablette tactile
- casques de réalité virtuelle, neuronaux ...



Ce que nous allons étudier

- Comment réaliser des IHM classiques client lourd
 - Interaction : souris, clavier
 - Affichage sur écran
- Placer différents composants graphiques sur l'écran
- Gérer les interactions du logiciel avec l'utilisateur
- Développer des IHM en respectant le modèle d'architecture MVC (Modèle, Vue, Contrôleur)
- Introduction à la programmation réactive (binding)



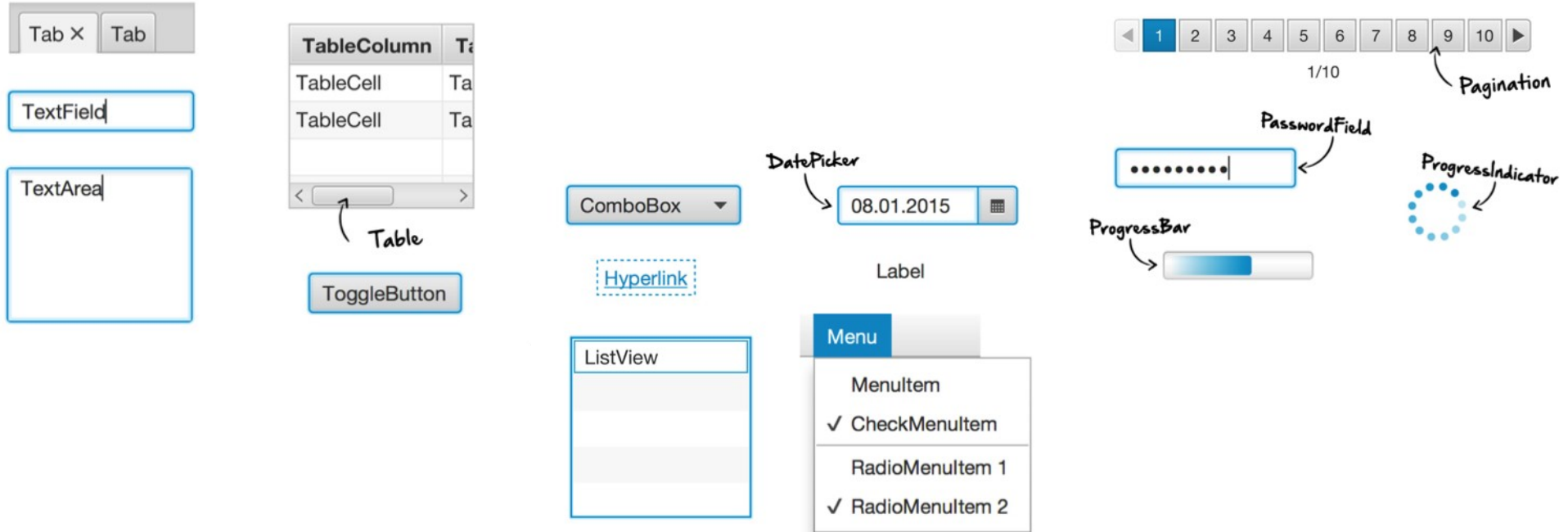
Développement avec JavaFX

- Il n'existe pas de bibliothèque spécifique pour développer des IHM client lourd en Kotlin (développement android oui).
- On va utiliser une bibliothèque java => **JavaFX**
<https://docs.oracle.com/javase/8/javafx/api/toc.htm>
- TornadoFX est une transcription de **JavaFX** en **Kotlin** mais difficile à appréhender en première année.
- JavaFX est une bibliothèque qui a beaucoup évolué donc **attention**, risque d'utiliser des ressources sur Internet qui sont obsolètes
- La plupart des documentations, exemples que vous trouverez concerneront java. Il faudra traduire en Kotlin.

Spécificités de JavaFX

- **JavaFX** (créé en 2008) est depuis 2014 la bibliothèque officielle pour créer des Interfaces Hommes Machines (IHM) avec Java
- Depuis Java 11 la bibliothèque **JavaFX** ne fait plus partie du jdk (Java Development Kit)
- **JavaFX** est une bibliothèque très riche qui comprend de nombreux mécanismes et composants que l'on retrouve dans la plupart des boîtes à outils dédiées au développement d'IHM :
 - => Les widgets ou controls (composants graphiques interactifs, comme un bouton)
 - => Les conteneurs qui permettent de construire et de structurer une IHM
 - => La programmation par événements qui permet à une application de réagir aux actions d'un utilisateur
- Possibilité de créer des interfaces soit en programmant ou soit en décrivant l'interface en xml => nous utiliserons le premier mode de création

Quelques widgets ou controls



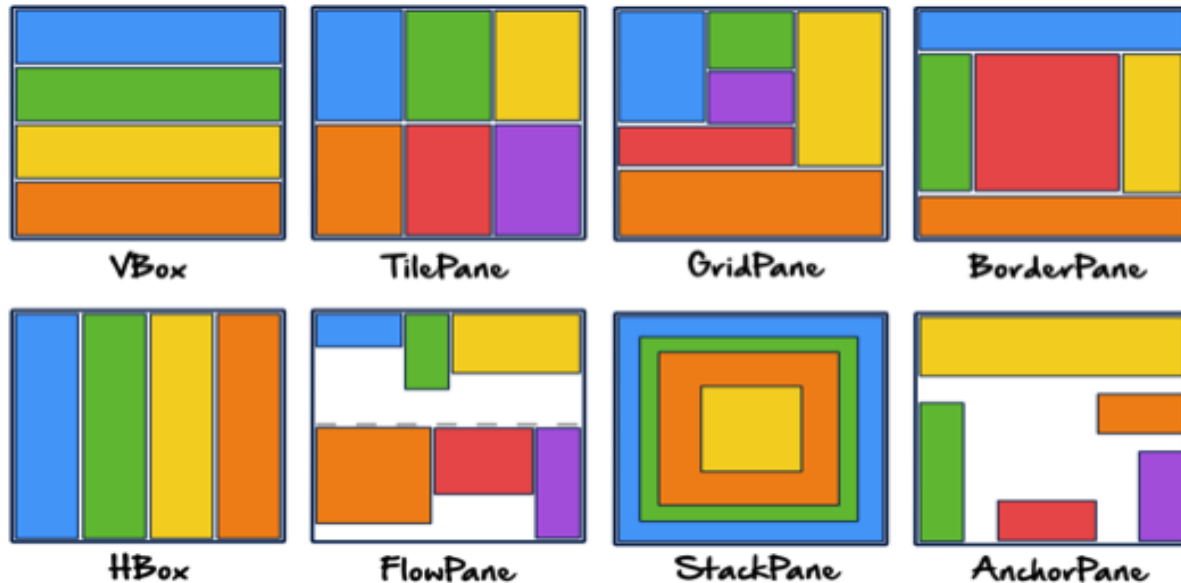
Les conteneurs (1)

Les conteneurs jouent un rôle important dans la structuration et la disposition des composants qui seront placés dans les interfaces. On s'intéresse à:

- la taille des composants
- à la position des composants
 - dans le conteneur
 - relative les uns par rapport aux autres
- aux alignements et espacements qui favorisent la structuration visuelle
- aux bordures et aux marges (notamment autour des conteneurs)
- au comportement dynamique de l'interface lorsqu'on redimensionne la fenêtre, etc.

Les conteneurs (2)

Développer une interface consistera donc dans un premier temps à définir les composants graphiques et les conteneurs à utiliser pour satisfaire les besoins de l'utilisateur



Exemple d'interface

TD5

Ma super Bibliothèque

liste des livres

L'Appel de Cthulhu
Les lames du Cardinal
Mortimer
Procrastination
Les ch'tits hommes libres
La longue terre
Hyperion
L'echiquier du mal
Gagner la guerre

information Livre

Numéro:

4

Titre:

Procrastination

Catégorie:

Fantasy

Auteur:

Terry Pratchett

<

>

Modification livre

Ajout livre

Suppression livre

Choix des conteneurs

TD5

Ma super Bibliothèque

liste des livres

L'Appel de Cthulhu
Les lames du Cardinal
Mortimer
Procrastination
Les ch'tits hommes libres
La longue terre
Hyperion
L'echiquier du mal
Gagner la guerre

information Livre

Numéro: 4

Titre: Procrastination

Catégorie: Fantasy

Auteur: Terry Pratchett

<

>

Modification livre

Ajout livre

Suppression livre

Interagir avec l'utilisateur

Ensuite, on ajoute les gestionnaires d'événements qui permettent de réagir aux interactions avec l'utilisateur

TD5

Ma super Bibliothèque

liste des livres

- L'Appel de Cthulhu
- Les lames du Cardinal
- Mortimer
- Procrastination**
- Les ch'tits hommes libres
- La longue terre
- Hyperion
- L'echiquier du mal
- Gagner la guerre

information Livre

Numéro: 4

Titre: Procrastination

Catégorie: Fantasy

Auteur: Terry Pratchett

< >

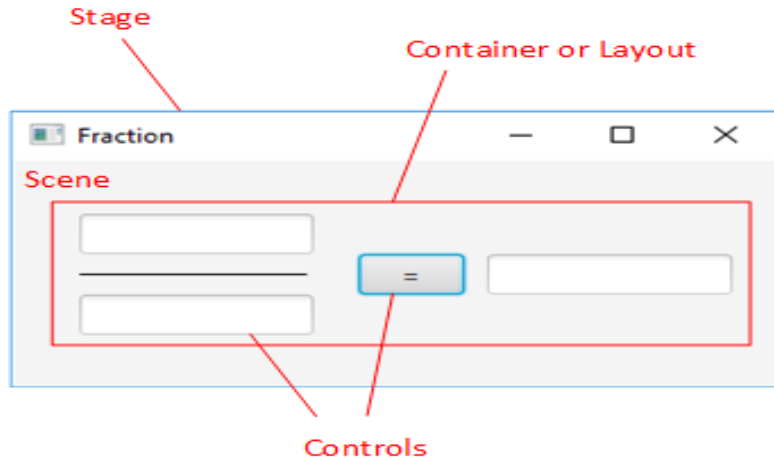
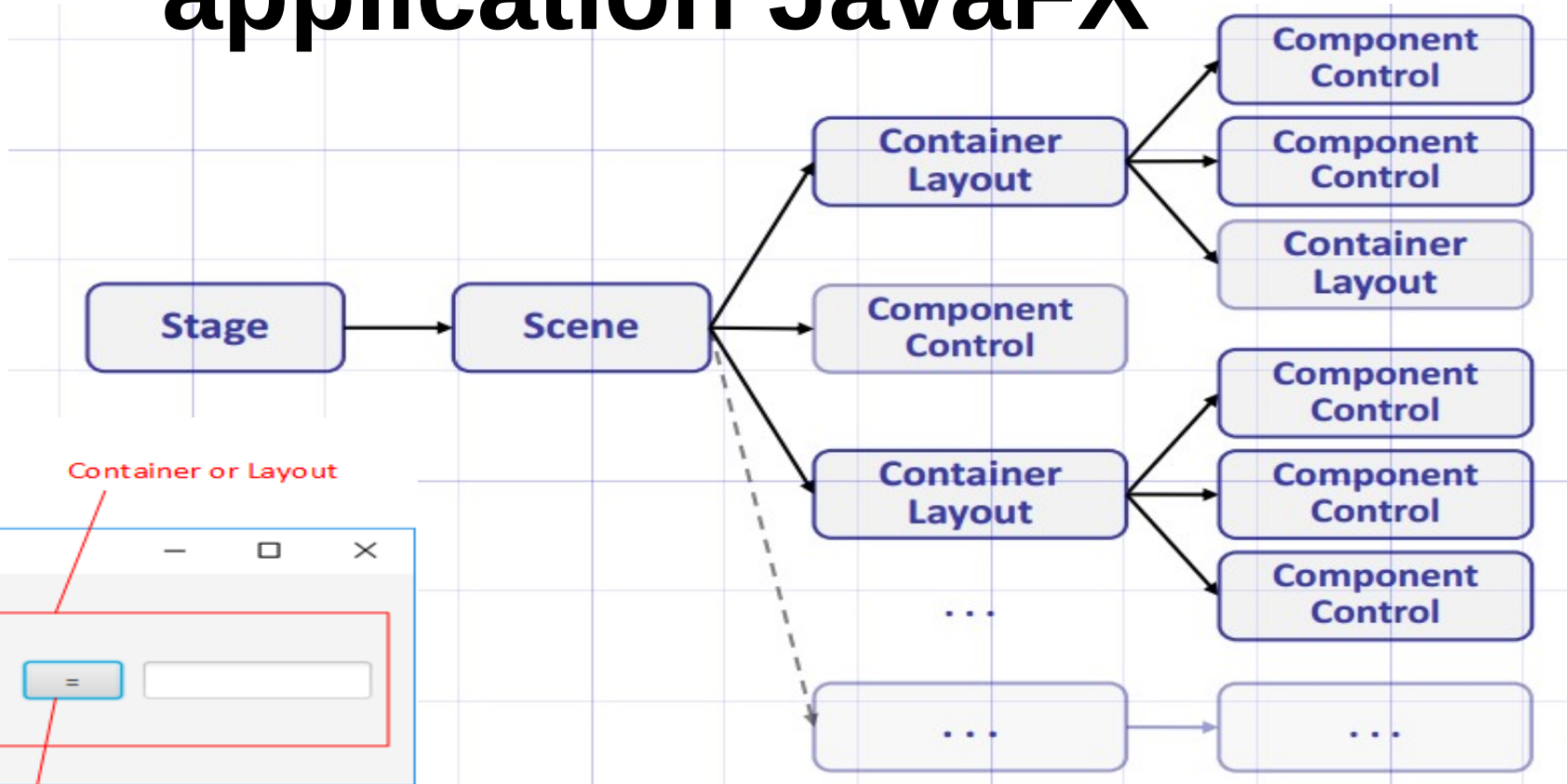
Modification livre Ajout livre Suppression livre

Si le bouton est cliqué, le livre précédent s'affiche et est souligné en bleu dans la liste

Si le bouton est cliqué, le livre suivant s'affiche et est souligné en bleu dans la liste

Si le bouton est cliqué, une fenêtre de Dialogue s'affiche pour confirmer la suppression

Structure générale d'une application JavaFX



Stage

Les conteneurs

Scene

Un
conteneur

TD5

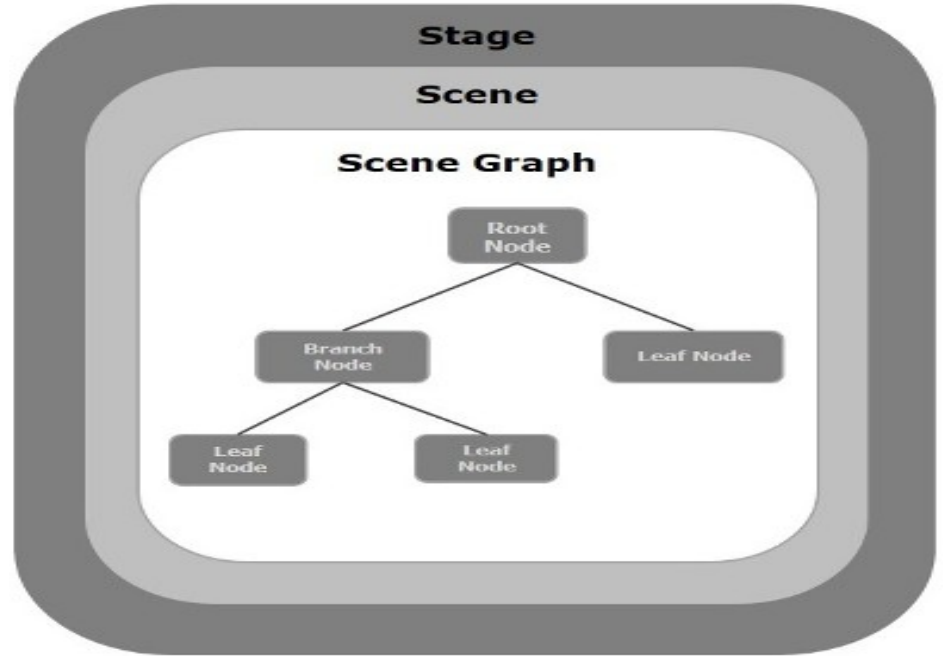
Ma super Bibliothèque

liste des livres	information Livre
L'Appel de Cthulhu Les lames du Cardinal Mortimer Procrastination Les ch'tits hommes libres La longue terre Hyperion L'echiquier du mal Gagner la guerre	<p>Numéro: 4</p> <p>Titre: <input type="text" value="Procrastination"/></p> <p>Catégorie: <input type="text" value="Fantasy"/></p> <p>Auteur: <input type="text" value="Terry Pratchett"/></p> <p><input data-bbox="940 840 1012 888" type="button" value=" < "/> <input data-bbox="1549 840 1621 888" type="button" value=" > "/></p>

Un
composant

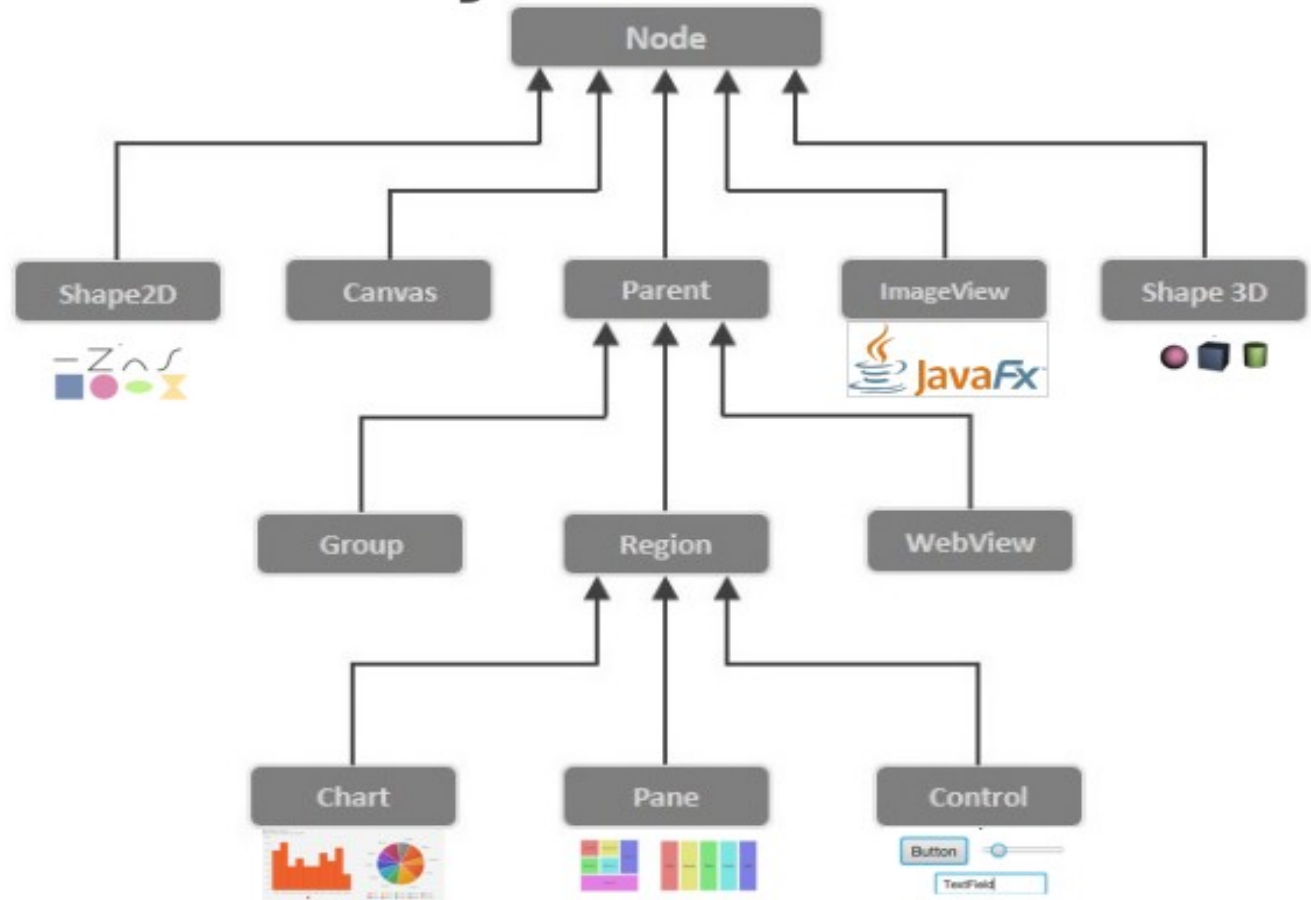
L'objet de type Scene

- L'objet de type **Scene** est un graphe qui est un arbre orienté qui représente la structure hiérarchique de l'interface.
- Les feuilles sont des **controls** (bouton, label, champs texte, ...)
- Les nœuds intermédiaires (y compris la racine) sont généralement des éléments de structuration => des conteneurs (HBox, VBox, BorderPane, ...)



Les Nodes

L'arbre est composé d'objet de type **Node**.



Une première application

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

```
class FirstAppli: Application() {
```

```
override fun start(primaryStage : Stage) {
    primaryStage.title="My First JavaFX App"
    val root = BorderPane()
    val btnHello = Button("Hello World")
    root.center=btnHello
    // contruction de la scène
    val scene = Scene(root, 250, 100)
    // association de la scène à stage
    primaryStage.scene=scene
    primaryStage.show()
}
```



```
// la fonction principale
fun main(args:Array<String>){
    Application.launch(FirstAppli::class.java)
}
```

Propriétés de la classe Region

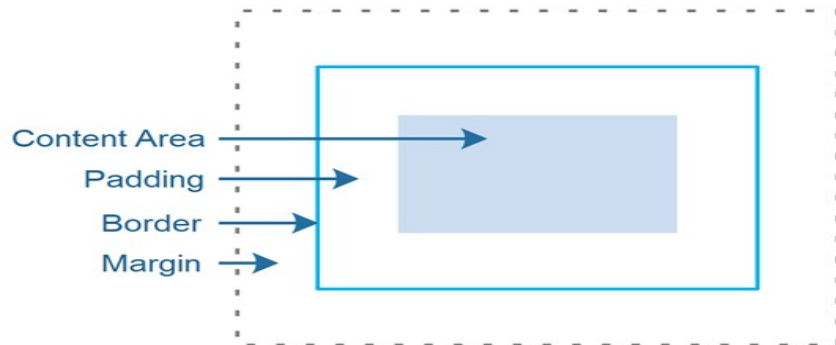
La classe **Region** est la classe parente des composants (Controls) et des conteneurs.

Les différentes zones d'une région sont basées sur la spécification du Box-Model CSS3 (W3C).

Voici quelques propriétés :

- **border** : bordure autour de la région
- **background** : couleur ou image d'arrière-plan de la région
- **padding** : espace entre le bord et le contenu de la région
- **margin** : espace entre la région et les autres composants dans le conteneur
- **style** : permet d'associer un style CSS à un noeud

```
nœud.style= "-fx-background-color : blue"
```



- En **JavaFX**, le style des composants est défini par des feuilles de style de type CSS.
- Il est ainsi possible de changer globalement l'aspect de l'interface sans avoir à modifier le code de l'application.

Les composants (controls)

- JavaFx est très riche en composants
- Ceux qui ont un libellé (**Labeled**) :
Label, ToggleButton, RadioButton, CheckBox, Button, Hyperlink
- Ceux qui contrôlent le texte saisi :
TextField, PasswordField, TextArea
- Ceux de type ComboBox :
ComboBox, ColorPicker, DatePicker
- Et plein d'autres : *ListView, TableView* ...
(voir diapo suivante)



Styleable
EventTarget

javafx.scene.Node

javafx.scene.Parent

javafx.scene.layout.Region

Skinnable

Control

#Control ()

Static Methods

List<CssMetaData> <? extends Styleable, ?>>
getClassCssMetaData ()

Property

ContextMenu get/set/contextMenuProperty ()
Tooltip get/set/tooltipProperty ()

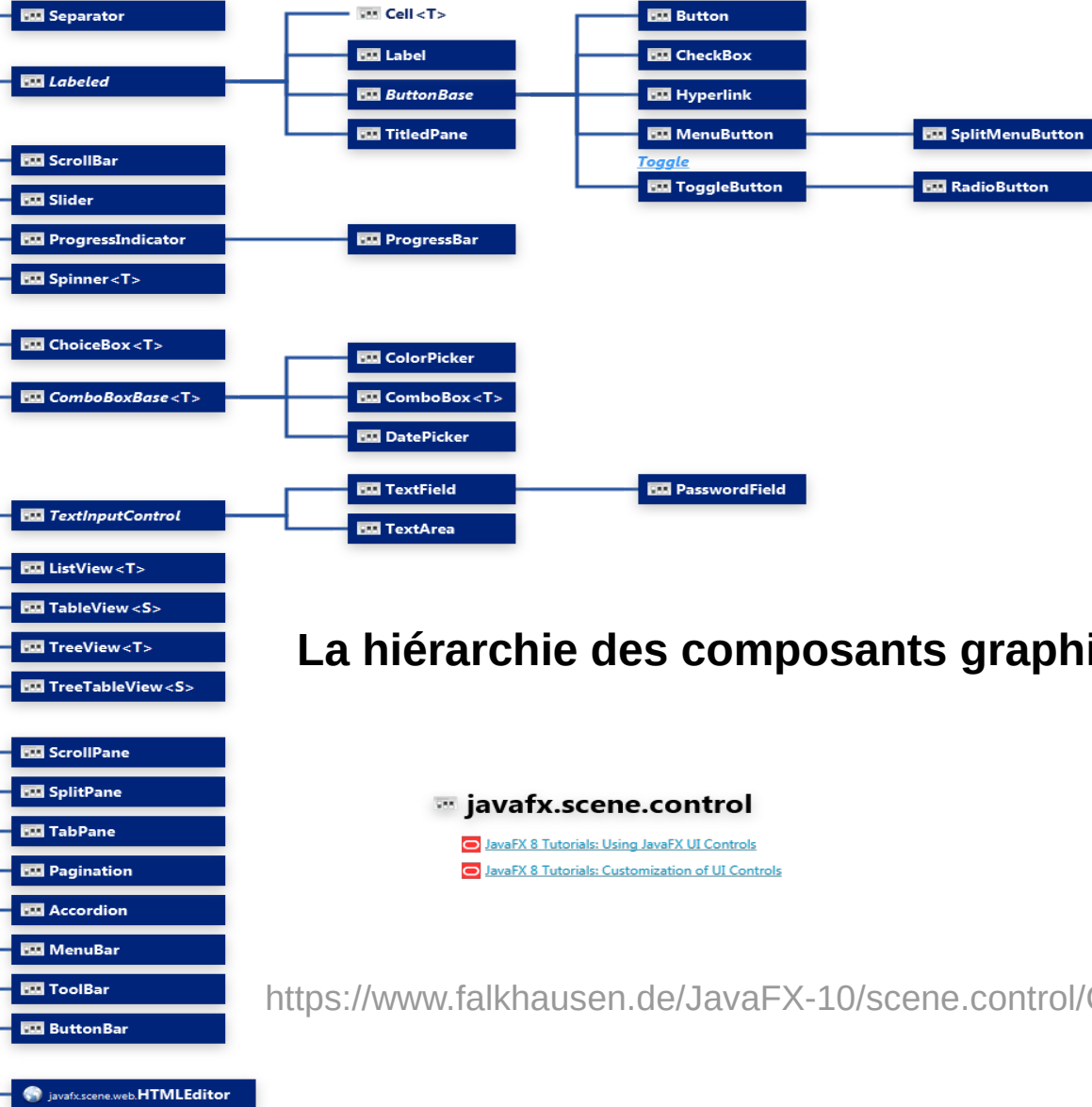
Accessor

#List<CssMetaData> <? extends Styleable, ?>>
getControlCssMetaData ()

Other Protected Methods

#Skin<?> createDefaultSkin ()

15 overriding - 3 deprecated methods hidden



La hiérarchie des composants graphiques (controls)

javafx.scene.control

JavaFX 8 Tutorials: Using JavaFX UI Controls

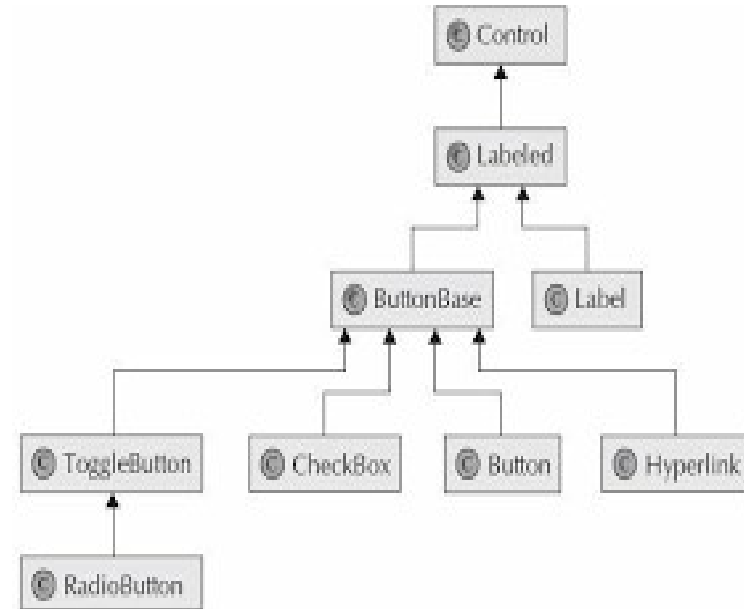
JavaFX 8 Tutorials: Customization of UI Controls

<https://www.falkhausen.de/JavaFX-10/scene.control/Control-Hierarchy.html>

La classe Labeled

Quelques propriétés des composants **Labeled** :

- **text** : le texte affiché
- **font** : type Font, police de caractères (style, taille, ...)
- **textFill** : couleur du texte
- **alignment** : type Pos, alignement général du texte (et du graphique éventuel) dans la zone
- **wrapText** : type Boolean, qui est vrai si le texte passe à la ligne suivante lorsqu'il atteint la limite de la zone.
- **textAlignment** : Type TextAlignment (LEFT, RIGHT, CENTER, JUSTIFY), alignement des lignes si le texte est multilignes.
- **lineSpacing** : type Double, définit l'espacement des lignes pour les textes multilignes
- **graphic** : image par exemple qui accompagne le texte
- **contentDisplay** : position du composant additionnel précédent par rapport au texte. Type énuméré ContentDisplay (LEFT, RIGHT, TOP, ...)

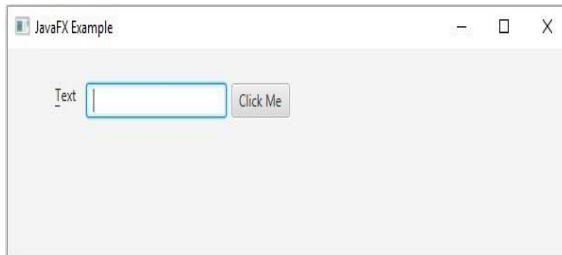
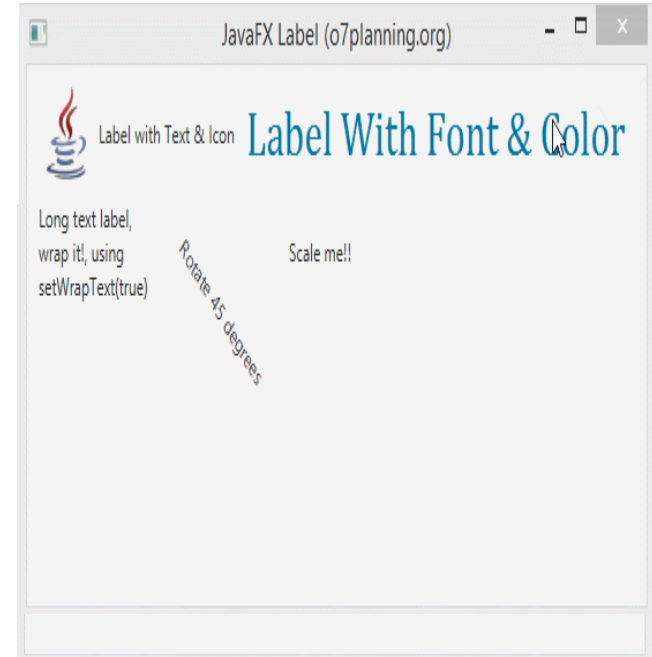


Exemple la classe Label

Un Label permet d'afficher du texte et/ou une image et n'est pas éditable

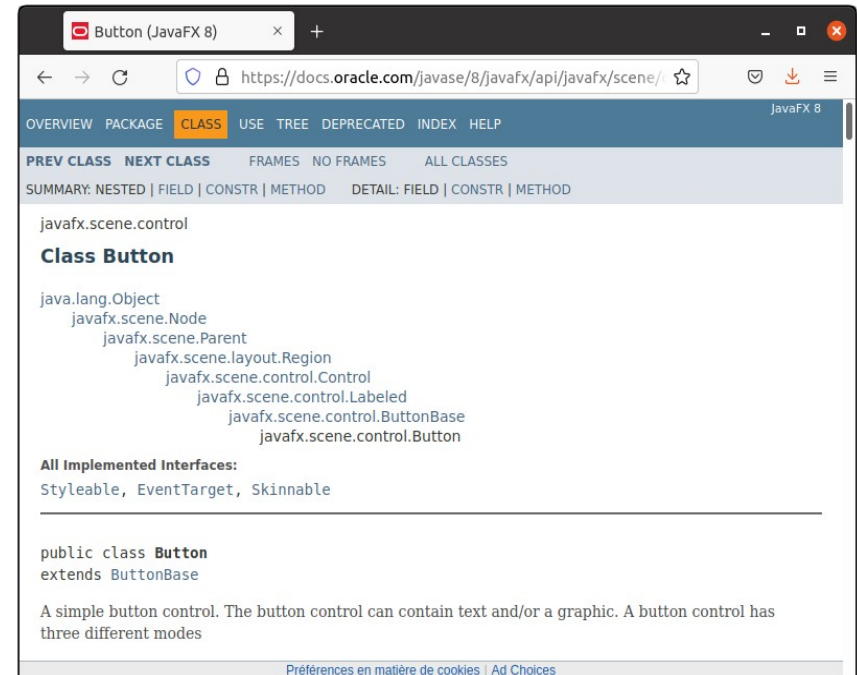
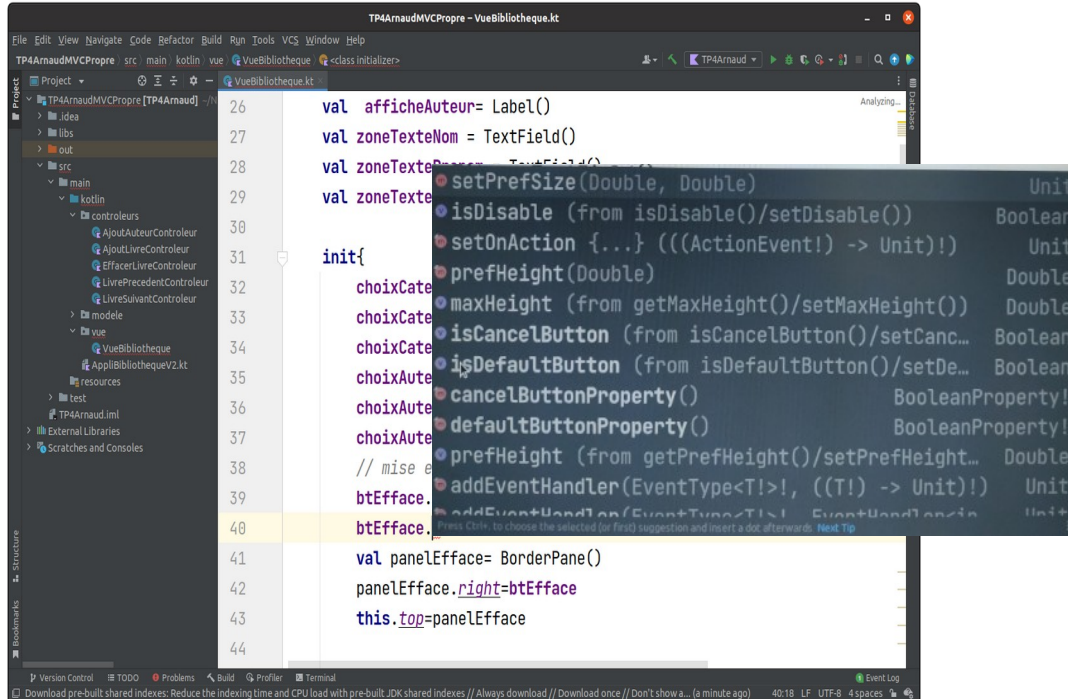
- Il hérite de toutes les propriétés de ses super classes et donc de la classe **Labeled**
- Une seule propriété additionnelle se trouve dans Label

labelFor : Permet de définir un (autre) composant (Node) auquel le libellé est associé

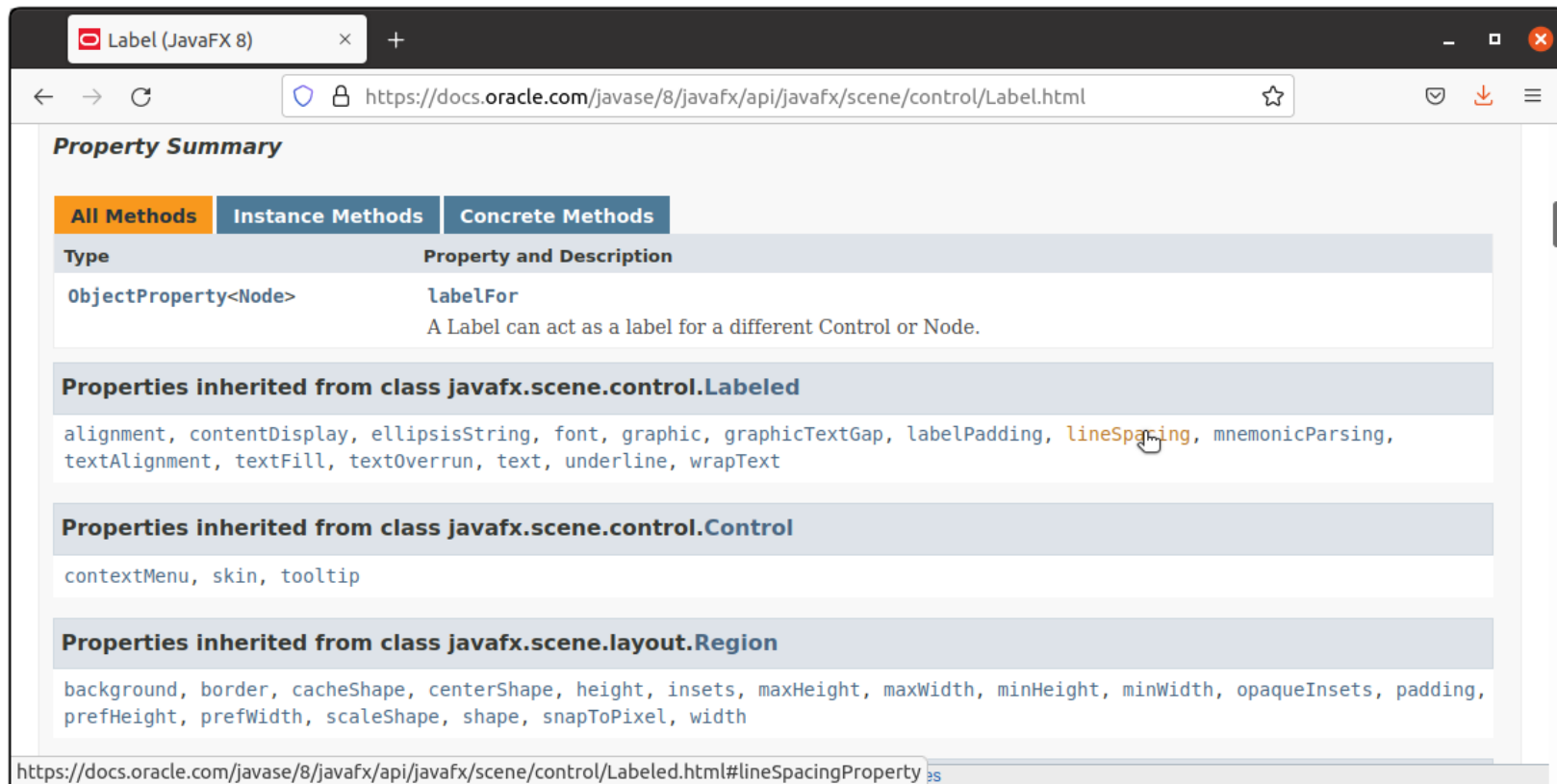


Javadoc et kotlin

- **JavaFX** est une **API java**, donc sa documentation (javadoc) a été rédigée pour le langage Java. Il vous faudra donc adapter, ce qui est lu, au langage **Kotlin**.
- **Java** heureusement est un langage proche de **Kotlin** et nous utiliserons l'IDE **IntelliJ** pour aider au développement



La javadoc de Label (1)



Label (JavaFX 8)

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Label.html

Property Summary

All Methods Instance Methods Concrete Methods

Type	Property and Description
<code>ObjectProperty<Node></code>	labelFor A Label can act as a label for a different Control or Node.

Properties inherited from class `javafx.scene.control.Labeled`

`alignment`, `contentDisplay`, `ellipsisString`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, **`lineSpacing`**, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`

Properties inherited from class `javafx.scene.control.Control`

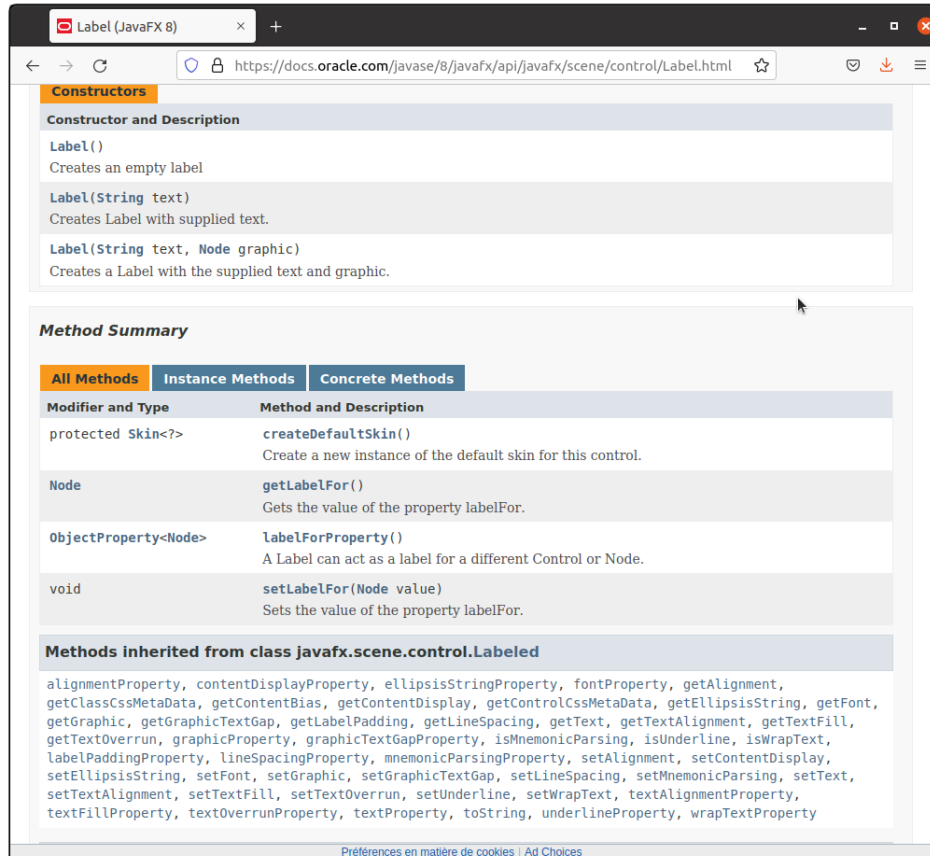
`contextMenu`, `skin`, `tooltip`

Properties inherited from class `javafx.scene.layout.Region`

`background`, `border`, `cacheShape`, `centerShape`, `height`, `insets`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `opaqueInsets`, `padding`, `prefHeight`, `prefWidth`, `scaleShape`, `shape`, `snapToPixel`, `width`

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Labeled.html#lineSpacingProperty

La Javadoc de Label (2)



The screenshot displays the Oracle Javadoc page for the `Label` class in JavaFX 8. The page is titled "Label (JavaFX 8)" and the URL is <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Label.html>.

Constructors

Constructor and Description

- `Label()`
Creates an empty label
- `Label(String text)`
Creates Label with supplied text.
- `Label(String text, Node graphic)`
Creates a Label with the supplied text and graphic.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
protected Skin<?>	<code>createDefaultSkin()</code> Create a new instance of the default skin for this control.
Node	<code>getLabelFor()</code> Gets the value of the property labelFor.
ObjectProperty<Node>	<code>labelForProperty()</code> A Label can act as a label for a different Control or Node.
void	<code>setLabelFor(Node value)</code> Sets the value of the property labelFor.

Methods Inherited from class `javafx.scene.control.Labeled`

`alignmentProperty, contentDisplayProperty, ellipsisStringProperty, fontProperty, getAlignment, getClassCssMetaData, getContentBias, getContentDisplay, getControlCssMetaData, getEllipsisString, getFont, getGraphic, getGraphicTextGap, getLabelPadding, getLineSpacing, getText, getTextAlignment, getTextFill, getTextOverrun, graphicProperty, graphicTextGapProperty, isMnemonicParsing, isUnderline, isWrapText, labelPaddingProperty, lineSpacingProperty, mnemonicParsingProperty, setAlignment, setContentDisplay, setEllipsisString, setFont, setGraphic, setGraphicTextGap, setLineSpacing, setMnemonicParsing, setText, setTextAlignment, setTextFill, setTextOverrun, setUnderline, setWrapText, textAlignmentProperty, textFillProperty, textOverrunProperty, textProperty, toString, underlineProperty, wrapTextProperty`

Préférences en matière de cookies | Ad Choices

Un exemple de code

```
val root=HBox()  
root.spacing=10.0  
val label1=Label("Hello")  
val label2=Label("Kotlin is fun !!")  
val label3=Label("Do you \nknow\nJava ?")  
label1.font=Font.font("Verdana", FontWeight.BOLD, FontPosture.REGULAR, 20.0)  
label2.font=Font("arial", 28.0)  
label2.textFill= Color.RED  
val input = FileInputStream("image/kotlin.png")  
val image = Image(input)  
label2.graphic=ImageView(image)  
label2.contentDisplay= ContentDisplay.BOTTOM  
root.children.add(label1)  
root.getChildren().add(label2)  
root.getChildren().add(label3)  
primaryStage.scene= Scene (root)  
primaryStage.show()
```

