



Ministère de l'Enseignement Supérieur et de la  
Recherche Scientifique et de la Technologie  
Institut Supérieur des  
Études Technologiques Bizerte



## Interface graphique avec JavaFX

# Bref Historique



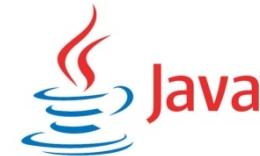
- ❑ A l'origine du langage Java , les interfaces graphiques étaient créées en utilisant la librairie AWT (java.awt)
  - ❑ Composants "lourds" (heavyweight)
  - ❑ Difficulté de créer des applications multiplateformes
- ❑ Rapidement, la librairie Swing (javax.swing) est venu compléter (et partiellement remplacer) la librairie AWT
  - ❑ Composants "légers"(lightweight) dessinés par la librairie;
  - ❑ Tout les composants de Swing exceptés JApplet, JDialog, JFrame, Jwindow sont des composants légers
  - ❑ Offre plus de composants
  - ❑ Créer des applications multiplateformes
- ❑ **JavaFX 1 a tenté, sans grand succès, de remplacer Swing**
  - ❑ Créer des interfaces graphiques pour toutes les sortes d'applications (mobiles, web, sur poste de travail, etc);
  - ❑ 2007-2011: il a été basé sur un langage script spécifique (*JavaFX Script*)

# Potentiel de JavaFx 8 (1)



- ❑ 2011-actuellement : elle est incluse par défaut dans Java et directement accessible via un IDE (Netbeans, Eclipse, JDeveloper, etc)
- ❑ Les caractéristiques principales de JavaFX 8 :
  - ❑ Abandon du langage de script;
  - ❑ Choix de deux modes : interfaces basées sur du code Java (API) et/ou sur un langage descriptif utilisant une syntaxe XML : **FXML**;
  - ❑ Création d'un outil interactif **Scene Builder** pour créer graphiquement des interfaces et générer automatiquement du code FXML;
  - ❑ Utilisation possible de feuilles de styles CSS pour adapter la présentation sans toucher au code (créer des thèmes, des skins, etc.)

## Potentiel de JavaFx 8 (2)



- ❑ Déléguer la conception graphique de l'interface à un spécialiste (UI designer) qui n'a pas l'obligation de connaître et maîtriser le langage Java
- ❑ Appliquer des **feuilles de style CSS** afin de renforcer encore cette **séparation** entre le design graphique et les traitements qui seront effectués à l'aide de code Java
- ❑ Différents **composants complexes** sont disponibles et permettent, avec un minimum d'effort, de créer des applications riches :
  - ❑ Effets visuels (ombrages, transitions, animations, ...)
  - ❑ Graphiques 2D (charts)
  - ❑ Images, audio, vidéo (media player), etc...

# Des interfaces déclaratives vs procédurales (1)



- La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces graphiques des applications :

## 1) Manière déclarative

- En décrivant l'interface dans un fichier FXML (syntaxe XML)
- L'utilitaire graphique Scene Builder facilite la création et la gestion des fichiers FXML
- L'interface peut être créée par un designer (sans connaissance Java, ou presque...)
- Séparation entre présentation et logique de l'application (MVC)

## 2) Manière procédurale

- Utilisation d'API pour construire l'interface avec du code Java
- Création et manipulation dynamique des interfaces



Il est possible de mélanger les deux techniques au sein d'une même application

# Des interfaces déclaratives vs procédurales (1)



- La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces graphiques des applications :

## 1) Manière déclarative

- En décrivant l'interface dans un fichier FXML (syntaxe XML)
- L'utilitaire graphique Scene Builder facilite la création et la gestion des fichiers FXML
- L'interface peut être créée par un designer (sans connaissance Java, ou presque...)
- Séparation entre présentation et logique de l'application (MVC)

## 2) Manière procédurale

- Utilisation d'API pour construire l'interface avec du code Java
- Création et manipulation dynamique des interfaces



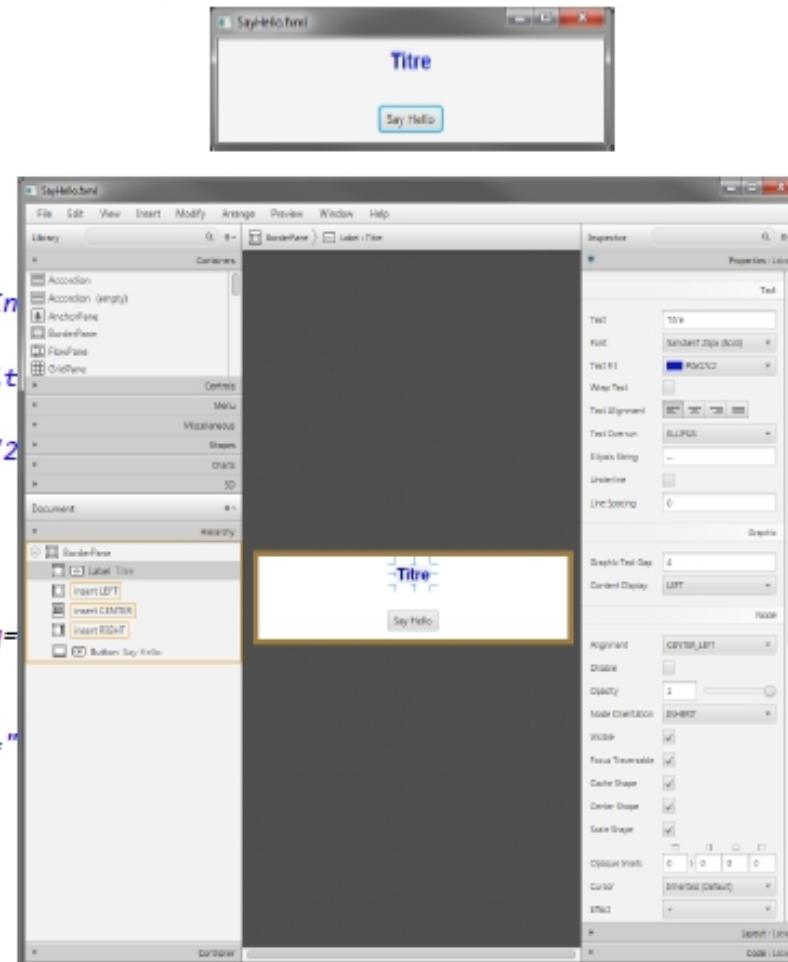
Il est possible de mélanger les deux techniques au sein d'une même application

# Des interfaces déclaratives vs procédurales (2)



- Technique déclarative (fichier FXML, Scene Builder et résultat).

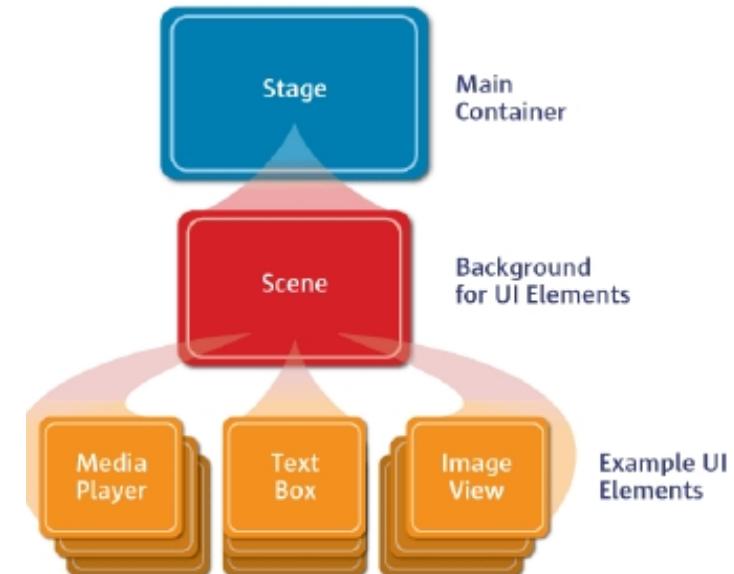
```
<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.*?>  
<?import javafx.scene.text.*?>  
<?import javafx.scene.control.*?>  
<?import java.lang.*?>  
<?import javafx.scene.layout.*?>  
  
<BorderPane maxHeight="-Infinity" maxWidth="-In  
    <top>  
        <Label id="title" fx:id="title" text="Tit  
            <font>  
                <Font name="SansSerif Bold" size="2  
            </font>  
        </Label>  
    </top>  
    <bottom>  
        <Button fx:id="btnHello" mnemonicParsing=  
    </bottom>  
    <padding>  
        <Insets bottom="10.0" left="10.0" right=""  
    </padding>  
</BorderPane>
```



# Manière procédurale : Concepts techniques



- Une application JavaFX est une classe qui doit hériter de la classe **Application** qui se trouve dans le package **javafx.application**;
- La fenêtre principale d'une application est représentée par un objet de type **Stage**;
- L'interface est représentée par un objet de type **Scene** qu'il faut créer et associer à la fenêtre (Stage);
- La scène est composée des différents éléments de l'interface graphique (composants de l'interface graphique) qui sont des objets de type **Node**;



# JavaFX : Métaphore de la salle de spectacle



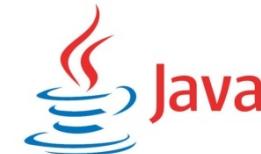
- ❑ **Stage** : L'endroit où a lieu l'action, où se déroule la scène
- ❑ **Scene**: Tableau ou séquence faisant intervenir les acteurs
- ❑ **Controls** : Acteurs, figurants, éléments du décor, ... qui font partie de la scène en train d'être jouée.
  - ❑ Components
  - ❑ Nodes
  - ❑ Widgets

# JavaFX : première application (1)



```
public class HelloWorld extends Application {  
    //----  
    @Override  
    public void start(Stage primaryStage) {  
        Button btn = new Button();  
        btn.setText("Say 'Hello World'");  
        StackPane root = new StackPane();  
        root.getChildren().add(btn);  
        Scene scene = new Scene(root, 300, 250);  
        primaryStage.setTitle("Hello World!");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    //----  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```





- ❑ La classe Application est une classe abstraite qui contient une méthode abstraite nommé « **start** »
  - ❑ Redéfinir la méthode « start »;
- ❑ La déclaration de la méthode « start » dans la classe « Application »  
**public abstract void start (Stage stage) throws java.lang.Exception**
- ❑ La redéfinition de la méthode « start » dans la classe « HelloFXApplication »

```
import javafx.application.Application;
import javafx.stage.Stage;
public class HelloFXApplication extends Application {
    public void start(Stage primaryStage) {
    }
}
```

L'estrade principale (en anglais **stage**)

## JavaFX : première application (3)



- Pour afficher l'estrade principale, lui définir **un titre, sa hauteur et sa largeur** : La déclaration de la méthode « start » dans la classe « Application »

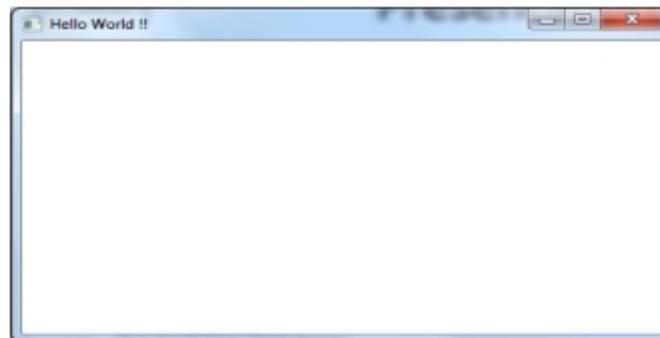
```
public void start(Stage primaryStage) {  
    primaryStage.setTitle("Hello World !!");  
    primaryStage.setHeight(300);  
    primaryStage.setWidth(500);  
    stage .show();  
}
```



### Sans main ()

Java qui lance l'application JavaFX si la classe en cours d'exécution ne contient pas la méthode main()

```
public class JavaFXApplication1 extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
  
        primaryStage.setTitle("Hello World !!");  
        primaryStage.setHeight(300);  
        primaryStage.setWidth(500);  
        primaryStage.show();  
  
    }  
}
```



### Avec main ()

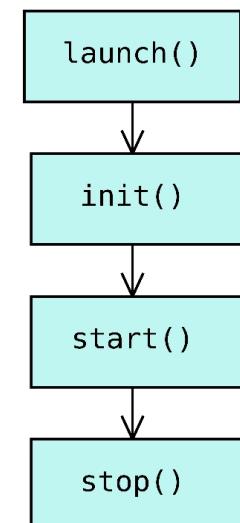
Il faut appeler la méthode `launch ()` de la classe Application

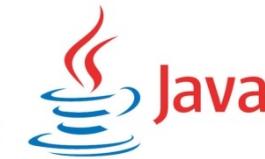
```
public class JavaFXApplication1 extends Application {  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
  
        primaryStage.setTitle("Hello World !!");  
        primaryStage.setHeight(300);  
        primaryStage.setWidth(500);  
        primaryStage.show();  
  
    }  
}
```

# Cycle de vie d'une application JavaFX

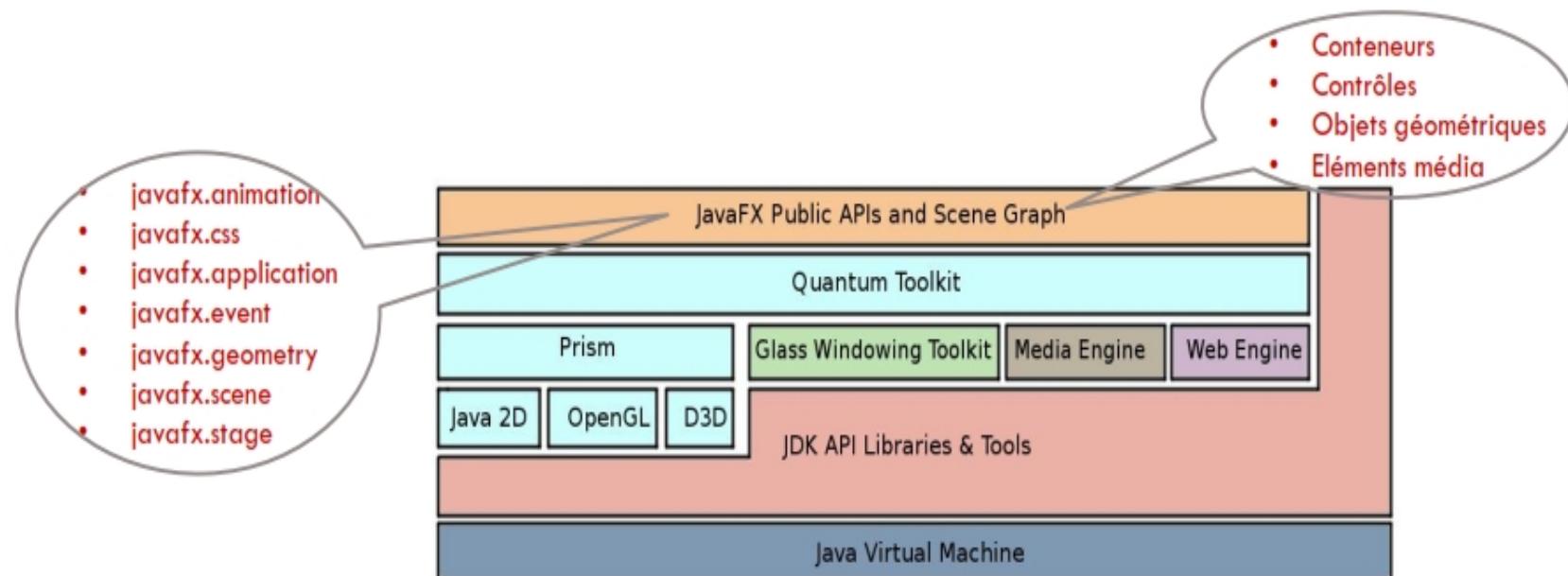


- ❑ Le point d'entrée d'une application JavaFX est constitué de l'instance de la classe Application (généralement une sous classe).
- ❑ Lors du lancement d'une application par la méthode statique **Application. Launch ()** le runtime JavaFX effectue les opérations suivantes :
  1. Crée une instance de la classe qui hérite de Application;
  2. Appelle la méthode **init()**
  3. Appelle la méthode **start()** et lui passe en paramètre une instance de Stage (qui représente la fenêtre principale [*primary stage*])
  4. Attend ensuite que l'application se termine; cela se produit lorsque :
    - La dernière fenêtre de l'application a été fermée
    - L'application appelle Platform.exit() (ne pas utiliser System.Exit())
  5. Appelle la méthode **stop()**

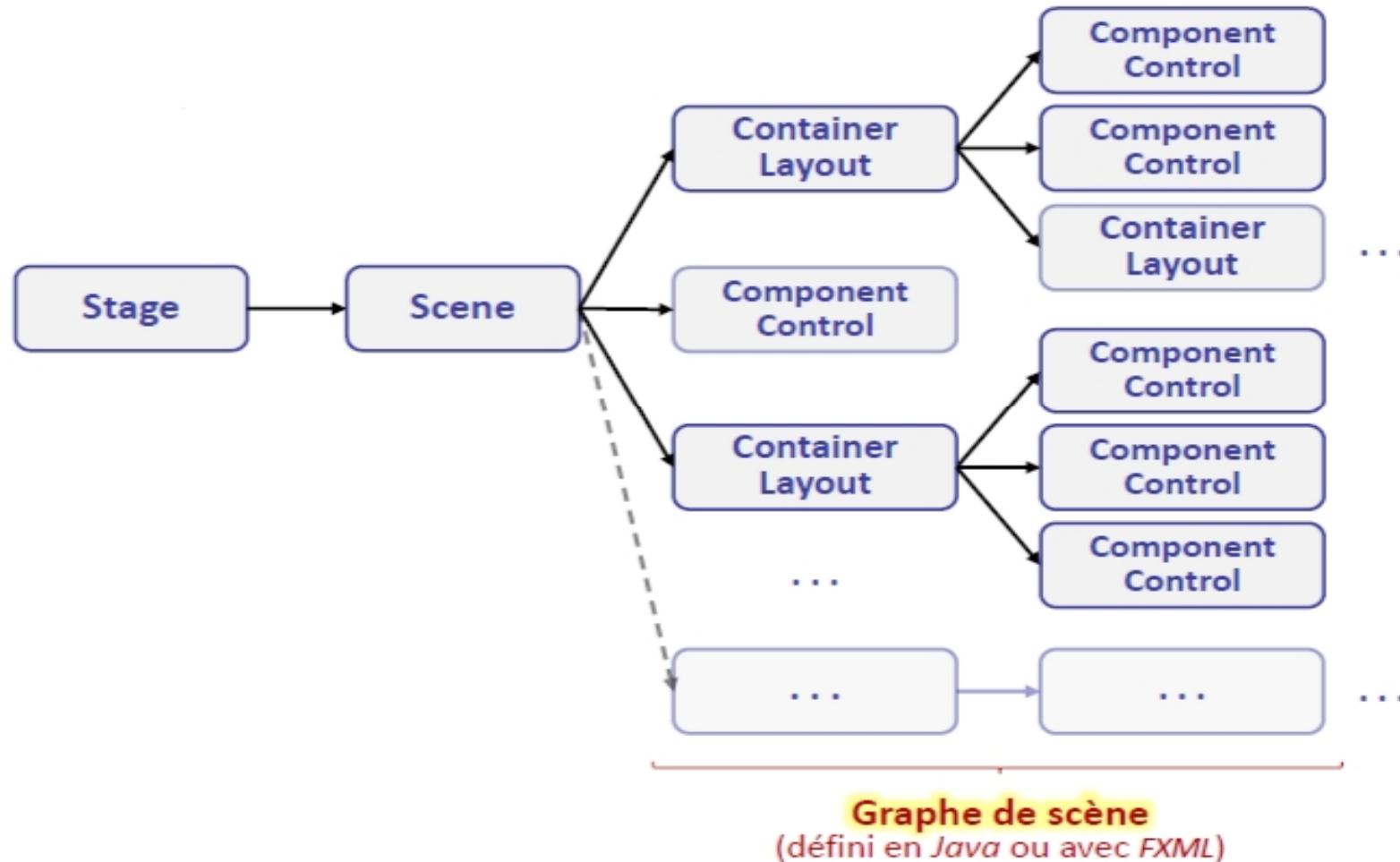




- ❑ L'architecture technique de la plateforme JavaFX est composée de plusieurs couches (library stack) qui reposent sur la machine virtuelle Java (JVM).
- ❑ Les développeurs ne devrait utiliser que la couche (API) publique car les couches inférieures sont susceptibles de subir des changements importants au fil des versions (sans aucune garantie de compatibilité)



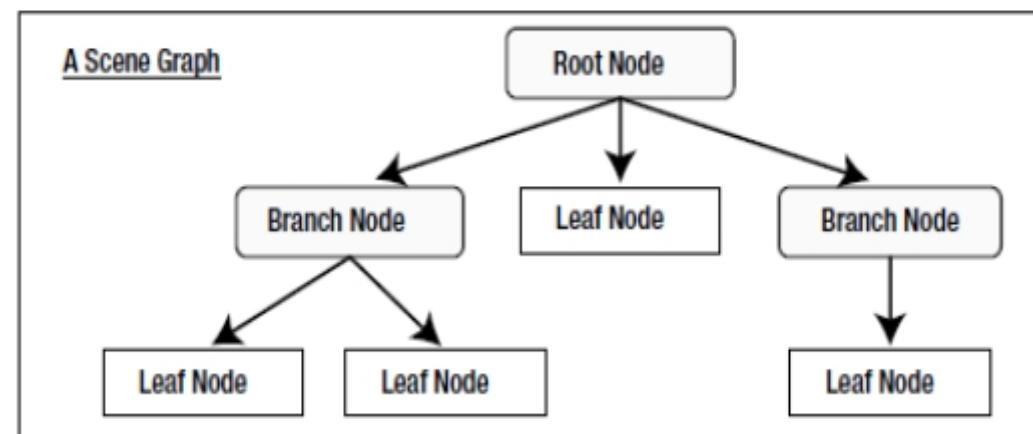
# Eléments d'une application JavaFX



## Graphe de scène (1)



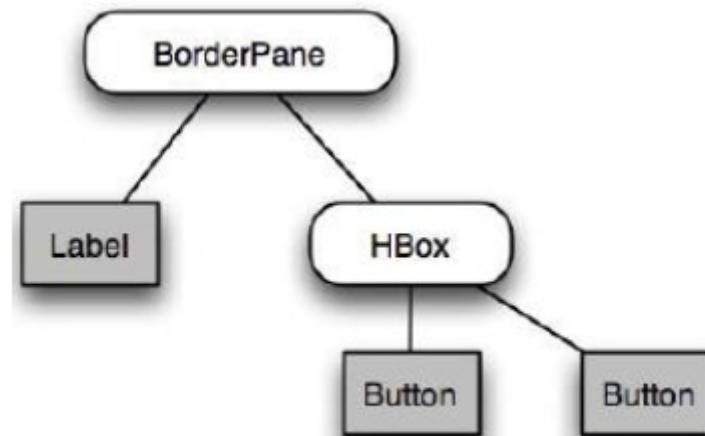
- Le **graphe de scène (scene graph)** est une notion importante qui représente la structure hiérarchique de l'interface graphique.
- Techniquement, c'est un graphe acyclique orienté (arbre orienté) avec :
  - une racine (root)
  - des noeuds (nodes)
  - des arcs qui représentent les relations parent-enfant
- Les noeuds (nodes) peuvent être de trois types :
  - Racine
  - Noeud intermédiaire
  - Feuille (leaf)



## Graphe de scène (2)



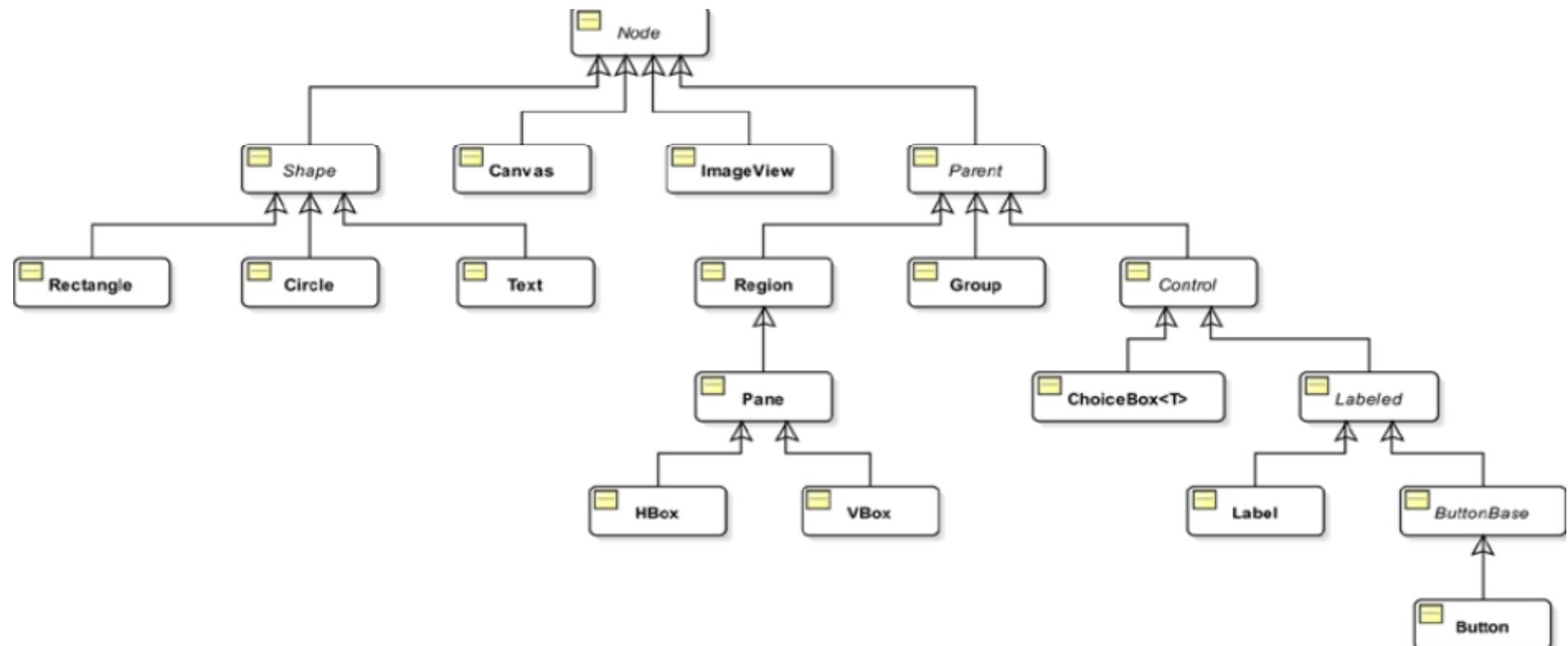
- Les feuilles de l'arbre sont généralement constitués de **composants visibles**(boutons, champs texte, ...)
- les nœuds intermédiaires et le nœud racine sont généralement des éléments de structuration (souvent **invisibles**), **typiquement des conteneurs** de différents types (HBox, VBox, BorderPane, ...).



## Graphe de scène (3)

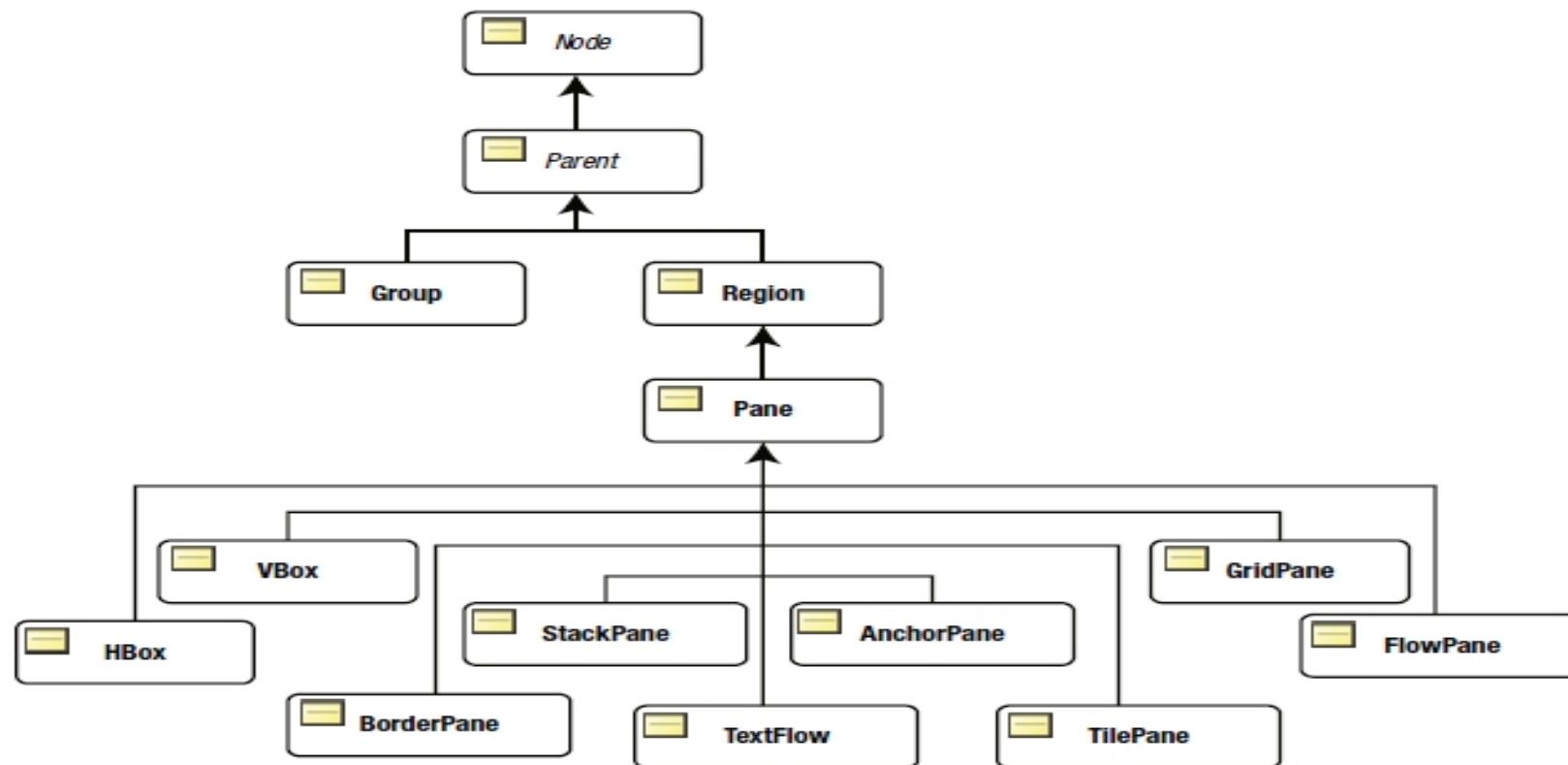


- Tous les éléments contenus dans un graphe de scène sont des objets qui ont pour classe parente la classe **Node**.
- La classe **Node** comporte de nombreuses **sous-classes** :





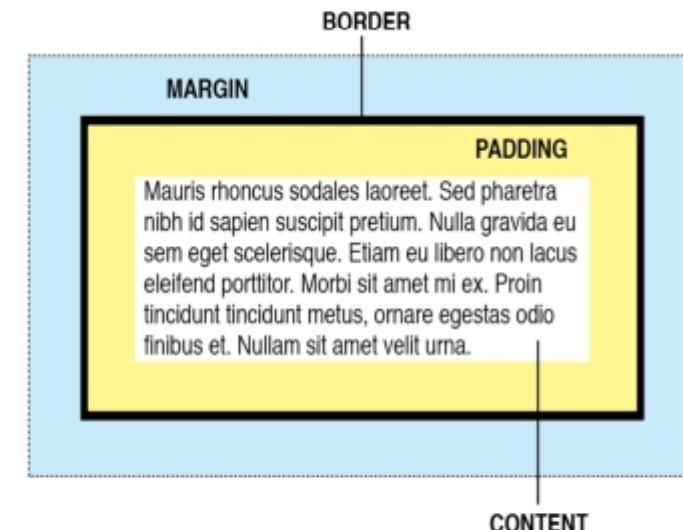
- Les **conteneurs (*Layout-Pane*)** représentent une famille importante parmi les sous-classes de Node. Ils ont pour classe parente **Pane** et **Region** qui possèdent de nombreuses propriétés et méthodes héritées par tous les conteneurs.





- ❑ La classe **Region** est la classe parente des composants (**Controls**) et des conteneurs (Layout-Panes)
- ❑ Elle définit des propriétés qui affectent la représentation visuelle.
- ❑ Elles définissent les notions **Margin, Border, Padding, Insets, Content**

- ❑ **Content Area** est la zone qui contient les composants enfants
  - ✓ Background : une couleur ou une image
- ❑ **Padding** : est un espace facultatif autour de la zone contenu
- ❑ **Border** est l'espace autour du padding
- ❑ **Margin** est l'espace situé autour de la bordure
- ❑ **Insets** : la classe Region a une propriété Padding de type Insets
  - ✓ Insets p=New Insets(5);
  - ✓ Insets p) New Insets (5,4, 3,2);

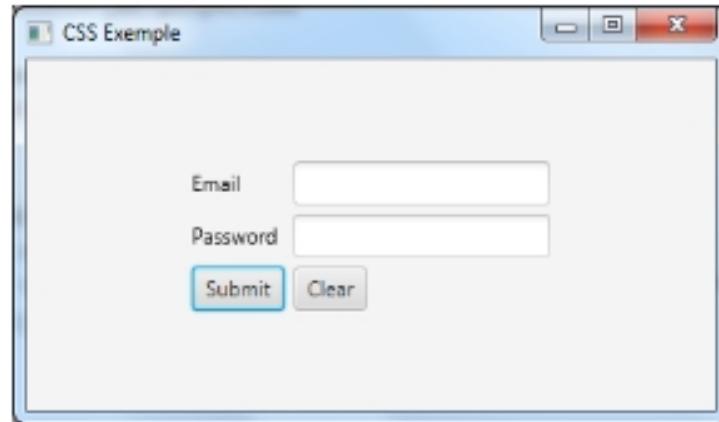


## L'apparence graphique : Look and feel(1)

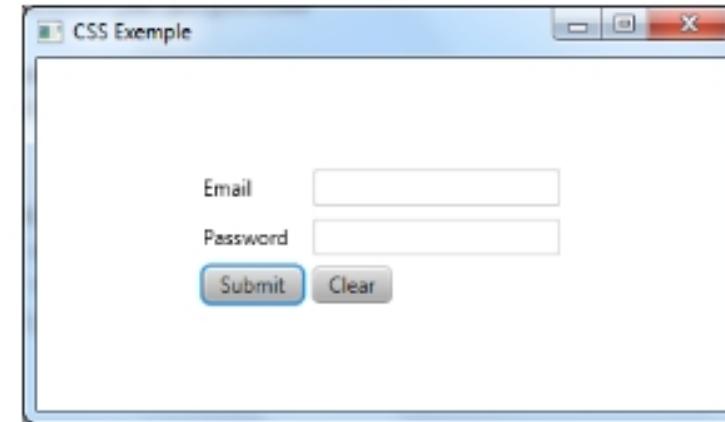


- La notion de **style**, **skin**, **thème** ou **look and feel (L&F)** caractérise l'ensemble des aspects visuels de l'interface graphique et de ses composants (forme, couleur, texture, ombre, police de caractères, ...).
- En *JavaFX*, le style des composants est défini par **des feuilles de style de type CSS**. Il est ainsi possible de changer globalement l'aspect de l'interface sans avoir à modifier le code de l'application.
- La méthode **setUserAgentStylesheet ()** de la classe Application permet d'indiquer l'URL de la feuille de style qui est à appliquer globalement.
- Deux styles, nommés **Modena** et **Caspian**, sont prédéfinis et sont associés aux constantes :
  - **STYLESHEET\_MODENA** : Utilisé par depuis *JavaFX 8*
  - **STYLESHEET\_CASPIAN** : A été défini pour *JavaFX 2*

## L'apparence graphique : Look and feel(2)



Le style : MODENA



Le style : CASPIAN

- Le style utilisé par défaut peut évoluer au fil des versions de *JavaFX*
- Si l'on veut fixer ou changer le *look and feel* des interfaces, on peut le faire au démarrage de l'application :

```
public void start(Stage primaryStage) {  
    ...  
    setUserAgentStylesheet(STYLESHEET_CASPIAN);  
    ...  
}
```

# L'apparence graphique : Look and feel(3)



Création de notre propre Style :

```
scene.getStylesheets().add(getClass().getResource("Style.css").toExternalForm());
```

Fichier  
Style.css

```
| .button {  
|     -fx-text-fill: white;  
|     -fx-font-family: "Arial Narrow";  
|     -fx-font-weight: bold;  
|     -fx-background-color: linear-gradient(#61a2b1, #2A5058);  
|     -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );  
|     /*définir des coins arrondis pour la bordure d'un élément*/  
|     -fx-background-radius: 16px;  
| }  
| .text-field {  
|     -fx-background-color:white ;  
|     -fx-background-insets: 0, 0 0 1 0 ;  
|     -fx-background-radius: 15px ;  
| }  
| .GridPanel{  
|     -fx-spacing: 10;  
|     -fx-alignment: center;  
|     -fx-background-color: red;  
|     -fx-background-image:url(im2.jpg);  
|     -fx-grid-lines-visible: true;  
| }  
| .Text {  
|     -fx-text-fill: white;  
| }
```

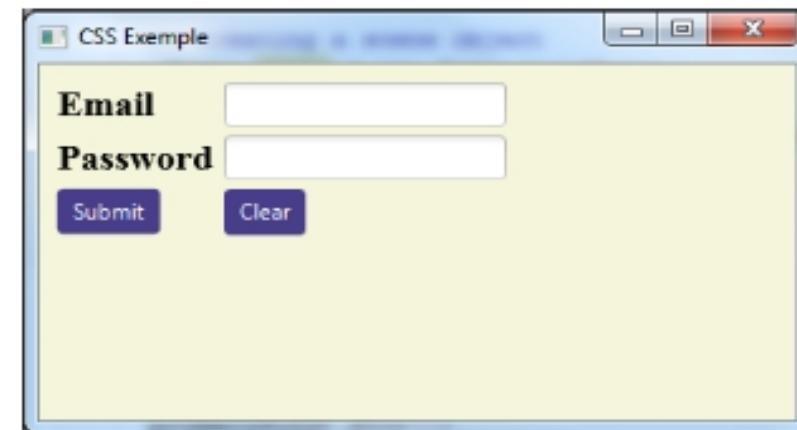


## L'apparence graphique : Look and feel(4)

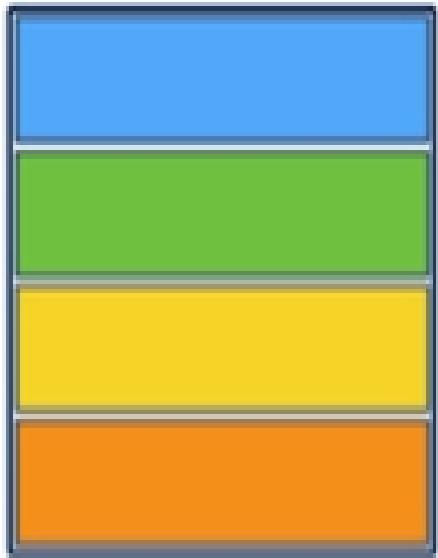


Création de notre propre Style :

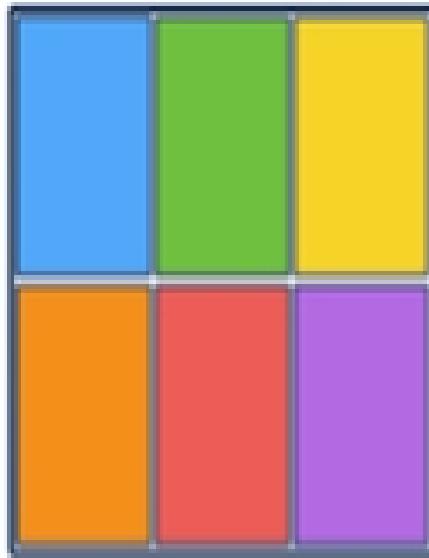
```
//Styling nodes
button1.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");
button2.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");
text1.setStyle("-fx-font: normal bold 20px 'serif' ");
text2.setStyle("-fx-font: normal bold 20px 'serif' ");
gridPane.setStyle("-fx-background-color: BEIGE;");
```



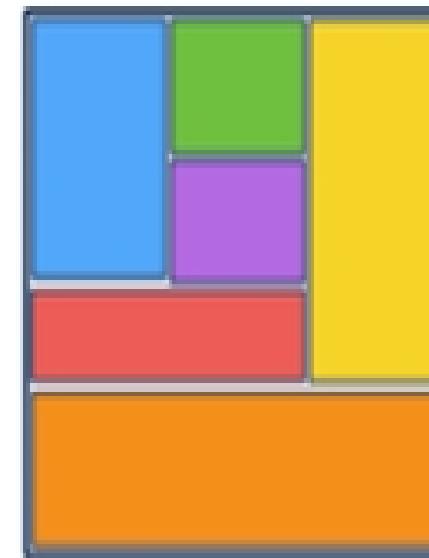
# les conteneurs



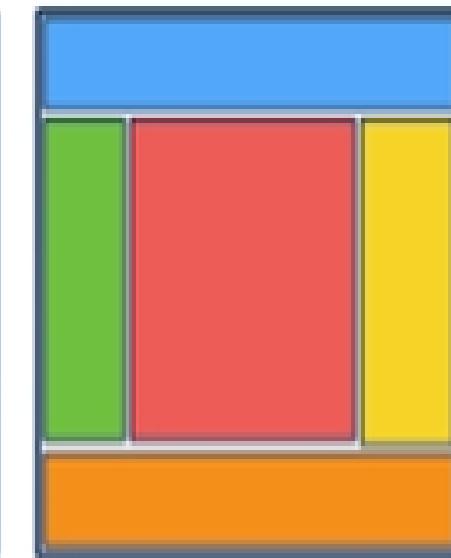
VBox



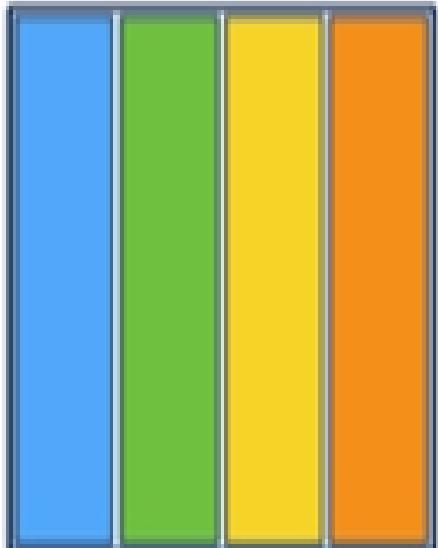
TilePane



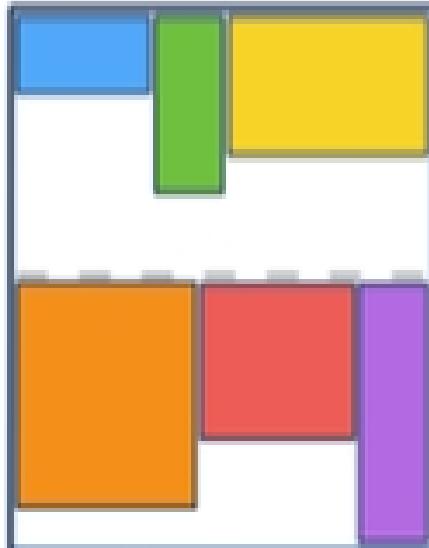
GridPane



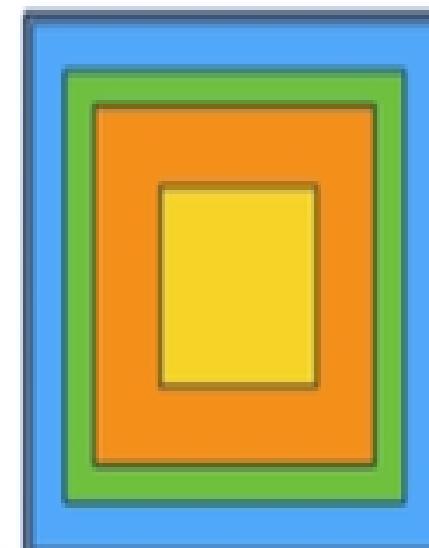
BorderPane



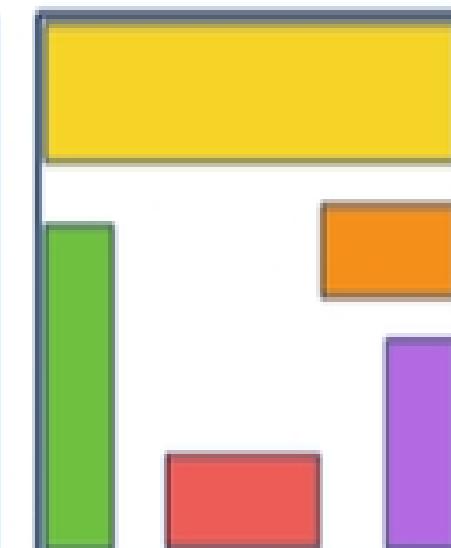
HBox



FlowPane

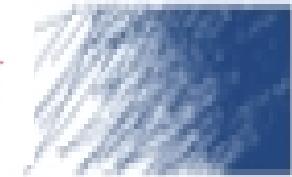


StackPane



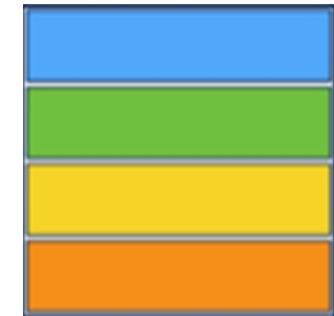
AnchorPane

# Le conteneur Vbox

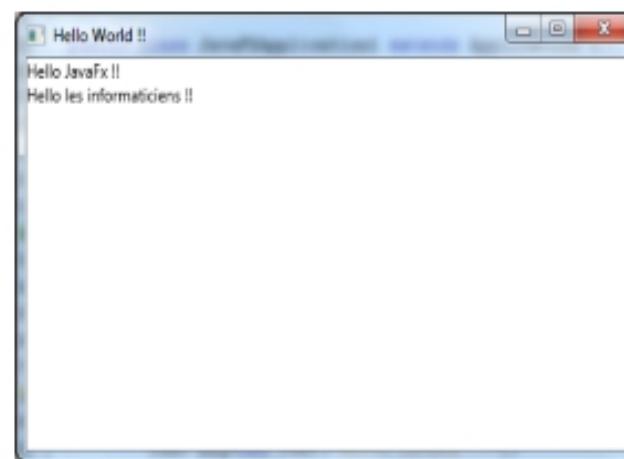


```
public void start(Stage primaryStage) {  
  
    primaryStage.setTitle("Hello World !!");  
    primaryStage.setHeight(300);  
    primaryStage.setWidth(500);  
  
    //Créer un noeud racine de type VBox  
    VBox root =new VBox();  
    //Créer un noeud texte  
    Text msg=new Text("Hello JavaFx !!");  
    //Créer un autre noeud texte  
    Text msg2=new Text("Hello les informaticiens !!");  
    //Ajouter le noeud texte au noeud racine  
    root.getChildren().add(msg);  
    root.getChildren().add(msg2);  
    //Créer une Scène  
    Scene scene=new Scene(root,300,500);  
    //Mettre la scène dans l'estrade  
    primaryStage.setScene(scene);  
  
    primaryStage.show();  
}
```

Vbox : (Vertical box): arrange les sous-composants sur une seule colonne;



VBox



# Le conteneur Hbox



```
public void start(Stage primaryStage) {  
  
    primaryStage.setTitle("Hello World !!");  
    primaryStage.setHeight(300);  
    primaryStage.setWidth(500);  
  
    //Créer un noeud racine de type HBox  
    HBox root =new HBox();  
    //Créer un noeud texte  
    Text msg=new Text("Hello JavaFx !!");  
    //Créer un autre noeud texte  
    Text msg2=new Text("Hello les informaticiens !!");  
    //Ajouter le noeud texte au noeud racine  
    root.getChildren().add(msg);  
    root.getChildren().add(msg2);  
    //Créer une Scène  
    Scene scene=new Scene(root,300,500);  
    //Mettre la scène dans l'estrade  
    primaryStage.setScene(scene);  
  
    primaryStage.show();  
  
}
```

Hbox : (Horizontal box): arrange les sous-composants sur une seule ligne;

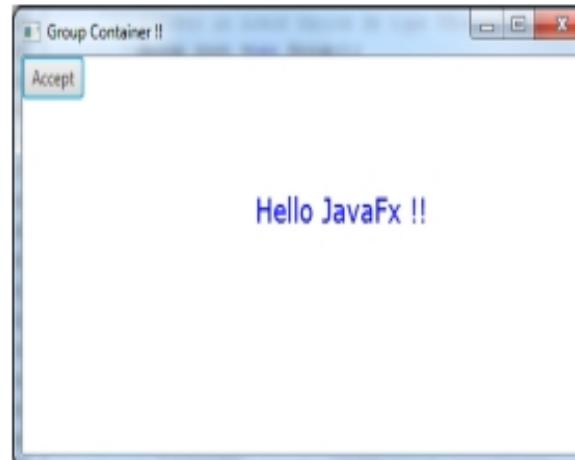


# Le conteneur Group

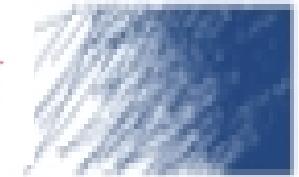


```
//Créer un noeud racine de type Group  
Group root =new Group();  
//Créer un noeud texte  
Text msg=new Text("Hello JavaFx !!");  
//Spécifier la police et la taille du texte  
msg.setFont(Font.font("Verdana", 20));  
//Spécifier la couleur du texte  
msg.setFill(Color.BLUE);  
//spécifier l'emplacement du texte  
msg.setX(200);  
msg.setY(100);  
//Créer un bouton Ok  
Button b1=new Button("Accept");  
//Ajouter le noeud texte au noeud racine  
root.getChildren().addAll(msg,b1);
```

**Group :** n'applique pas aucune disposition pour ses sous-composants  
Tous les sous-composants sont dans la position 0,0

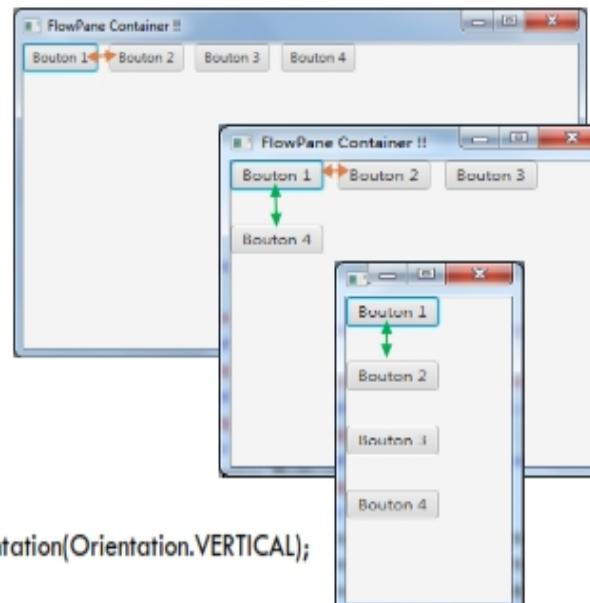


# Le conteneur FlowPane



```
//Créer un nœud racine de type FlowPane  
FlowPane root=new FlowPane();  
//l'écart horizontal entre les composants  
root.setHgap(10);  
//l'écart vertical entre les composants  
root.setVgap(30);  
//Créer les boutons  
Button b1=new Button("Bouton1");  
Button b2=new Button("Bouton2");  
Button b3=new Button("Bouton3");  
Button b4=new Button("Bouton4");  
//Ajouter les boutons au nœud racine  
root.getChildren().addAll(b1,b2,b3,b4);
```

**FlowPane** : organise les sous-composants horizontalement ou verticalement sur la même ligne ou sur la même colonne et si la ligne/colonne actuelle est remplie, elle passe à la ligne/colonne suivante.



root.setOrientation(Orientation.VERTICAL);



FlowPane

# Le conteneur TextFlow



```
TextFlow root=new TextFlow();
//l'alignement des objets Text dans le conteneur
root.setAlignment(TextAlignment.RIGHT);
//Définir l'espace entre les lignes de text
root.setLineSpacing(50.0);
```

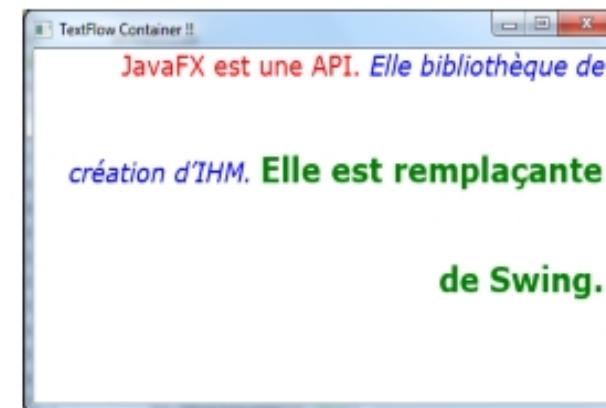
```
Text t1=new Text("JavaFX est une API. ");
t1.setFont(Font.font ("Verdana", 20));
t1.setFill(Color.RED);
```

```
Text t2=new Text("Elle bibliothèque de création d'IHM. ");
t2.setFont(Font.font ("Verdana",FontPosture.ITALIC, 20));
t2.setFill(Color.BLUE);
```

```
Text t3=new Text("Elle est remplaçante de Swing. ");
t3.setFont(Font.font ("Verdana",FontWeight.BOLD, 25));
t3.setFill(Color.GREEN);
```

```
root.getChildren().addAll(t1,t2,t3);
```

**Textflow :** ranger plusieurs noeuds text (CENTER, JUSTIFY, LEFT, RIGHT)

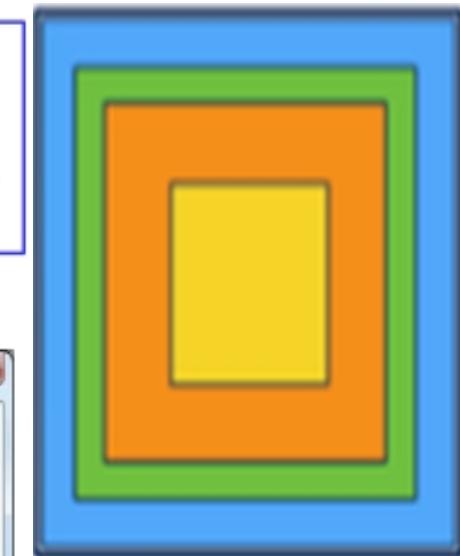
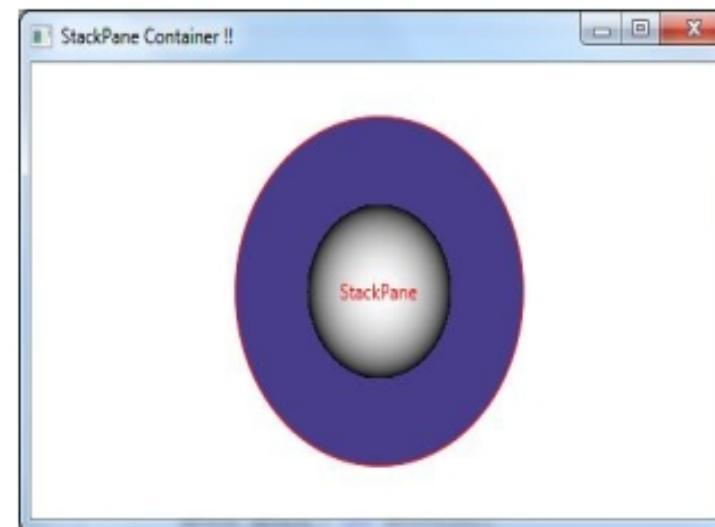


# Le conteneur StackPane



```
//Dessiner un cercle
Circle circle = new Circle(300, 195, 100);
//Couleur du Remplissage
circle.setFill(Color.DARKSLATEBLUE);
//Couleur du contour
circle.setStroke(Color.RED);
//Dessiner un sphère
Sphere sphere = new Sphere(50);
//Créer un texte
Text text = new Text("StackPane");
//Changer la couleur du sphère
text.setFill(Color.RED);
//Changer la position du texte
text.setX(20);
text.setY(50);
//Créer un Stack Pane
StackPane root = new StackPane();
//Récupérer la liste observable de la Stack Pane
ObservableList list = root.getChildren();
//Ajouter tous les noeuds au Stack Pane
list.addAll(circle, sphere, text);
// Créer une Scène
Scene scene=new Scene(root,100,100);
```

**StackPane :**  
Range les nœuds les uns  
sur les autres comme dans  
une pile



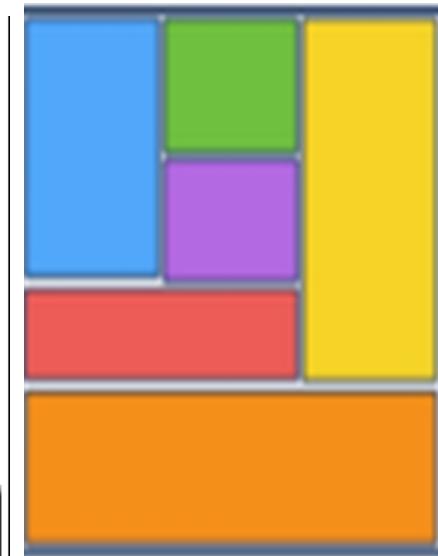
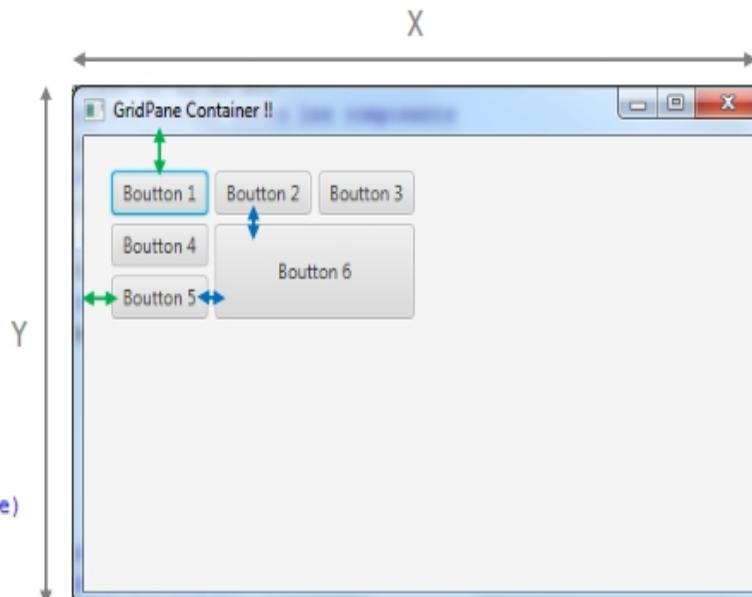
**StackPane**

# Le conteneur GridPane



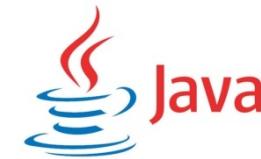
```
GridPane root=new GridPane();
//Buttons
Button b1=new Button("Boutton 1");
Button b2=new Button("Boutton 2");
Button b3=new Button("Boutton 3");
Button b4=new Button("Boutton 4");
Button b5=new Button("Boutton 5");
Button b6=new Button("Boutton 6");
//Alignment de la grille
root.setAlignment(Pos.TOP_LEFT);
//Noeud, indice des colonnes, indice des lignes
root.add(b1,0,0);
root.add(b2, 1, 0);
root.add(b3, 2, 0);
root.add(b4, 0, 1);
root.add(b5, 0, 2);
// Noeud, indice des colonnes, indice des lignes,
//largeur de la colonne, largeur de la ligne:
root.add(b6, 1, 1, 2, 2);
//L'ecart horizontal entre les composants
root.setHgap(5.0);  -->
//L'ecart vertical entre les composants
root.setVgap(5.0);  -->
//La marge autour de la grille (haut/droit/bas/gauche)
root.setPadding(new Insets(20, 20, 20, 20));  -->
//Agrandir la taille de b6
b6.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
```

GridPane :  
organiser ses sous-  
composants dans une  
grille.



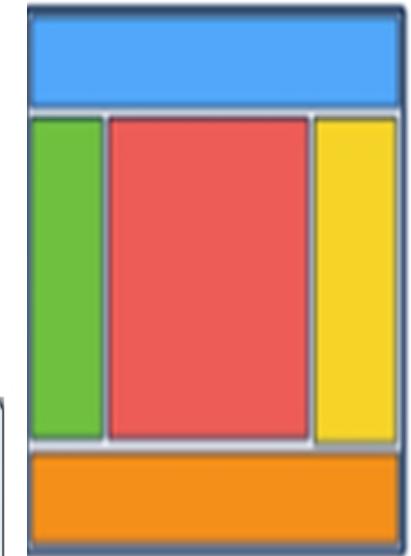
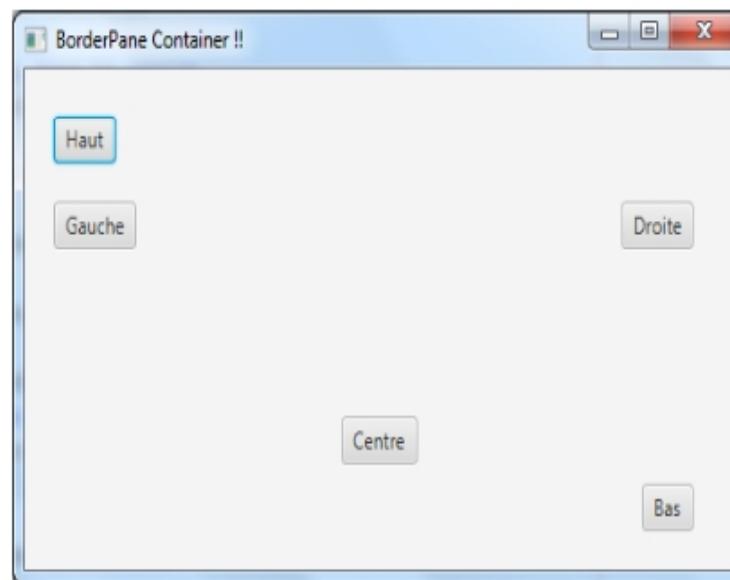
GridPane

# Le conteneur BorderPane



```
BorderPane root = new BorderPane();
//La marge autour le BorderPane
root.setPadding(new Insets(15, 20, 10, 10));
// Haut
Button bt1 = new Button("Haut");
root.setTop(bt1);
// Définir la marge pour la partie supérieur
BorderPane.setMargin(bt1, new Insets(10, 10, 10, 10));
// Gauche
Button bt2 = new Button("Gauche");
root.setLeft(bt2);
BorderPane.setMargin(bt2, new Insets(10, 10, 10, 10));
// Centre
Button bt3 = new Button("Centre");
root.setCenter(bt3);
// Alignment du boutton dans le centre
BorderPane.setAlignment(bt3, Pos.BOTTOM_CENTER);
// Droite
Button bt4 = new Button("Droite");
root.setRight(bt4);
BorderPane.setMargin(bt4, new Insets(10, 10, 10, 10));
// Bas
Button bt5 = new Button("Bas");
root.setBottom(bt5);
BorderPane.setMargin(bt5, new Insets(10, 10, 10, 10));
// Alignment du boutton dans zone base
BorderPane.setAlignment(bt5, Pos.TOP_RIGHT);
```

**BorderPane :**  
Est divisé en 5 zones distinctes, chacune d'elles peut contenir un sous-composant.



**BorderPane**

# Les composants de base : création et utilisation

## Les contrôles (1)

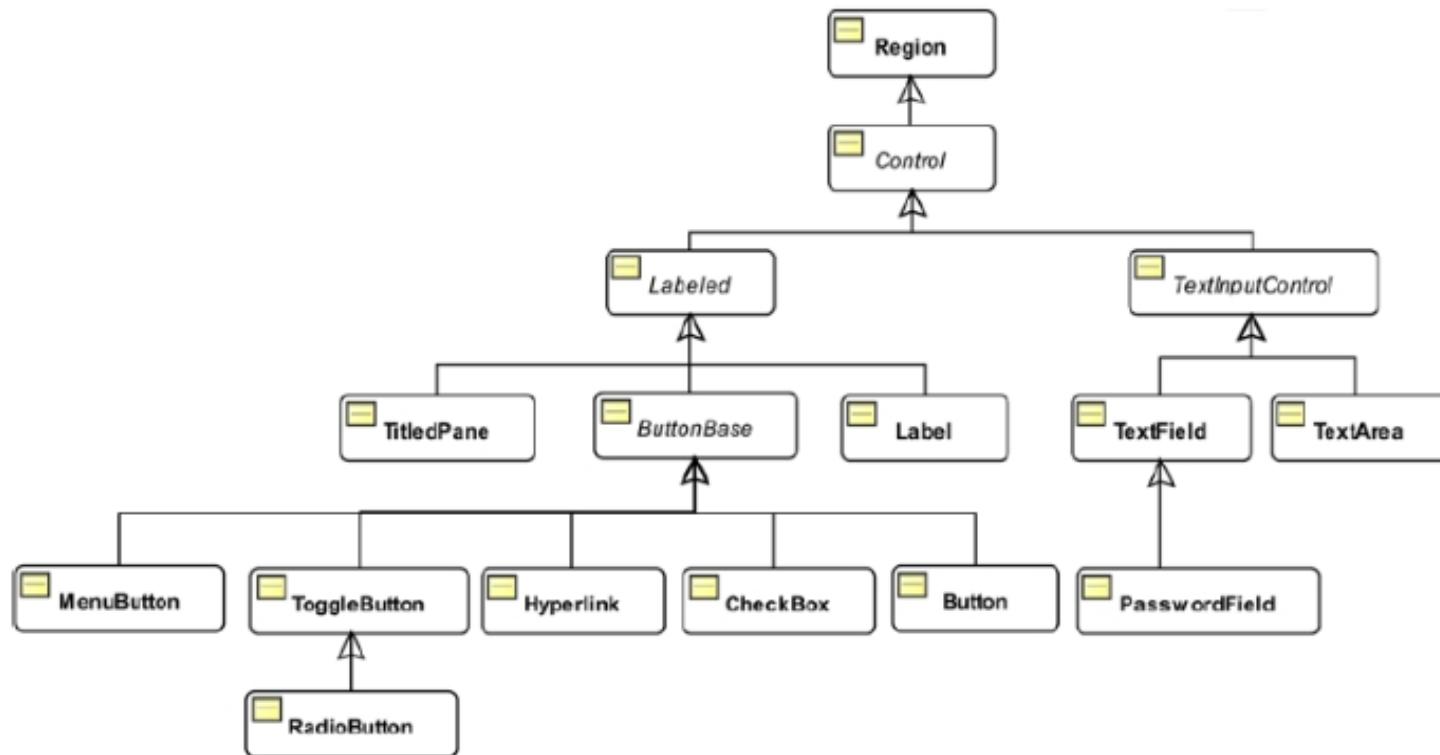


- ❑ JavaFX offre un ensemble de composants (kit de développement) pour créer les interfaces utilisateurs graphiques.
- ❑ Ces composants d'interface sont fréquemment nommés **controls** dans la documentation en anglais (parfois **widgets**).
- ❑ Dans ce cours, nous utiliserons généralement le terme composant pour parler des éléments qui servent à afficher des informations ou permettre à l'utilisateur d'interagir avec l'application.
  - ✓ Libellés, icônes, boutons, champs-texte, menus, cases à cocher, etc.
- ❑ Bien qu'ils constituent les deux des nœuds (**node**) du graphe de scène, les composants sont à distinguer des conteneurs (layout panes) qui servent à disposer les composants et qui ne sont pas directement visibles dans l'interface (les bordures et les couleurs d'arrière-plan permettent cependant de révéler leur présence).

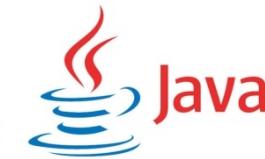
## Les contrôles (2)



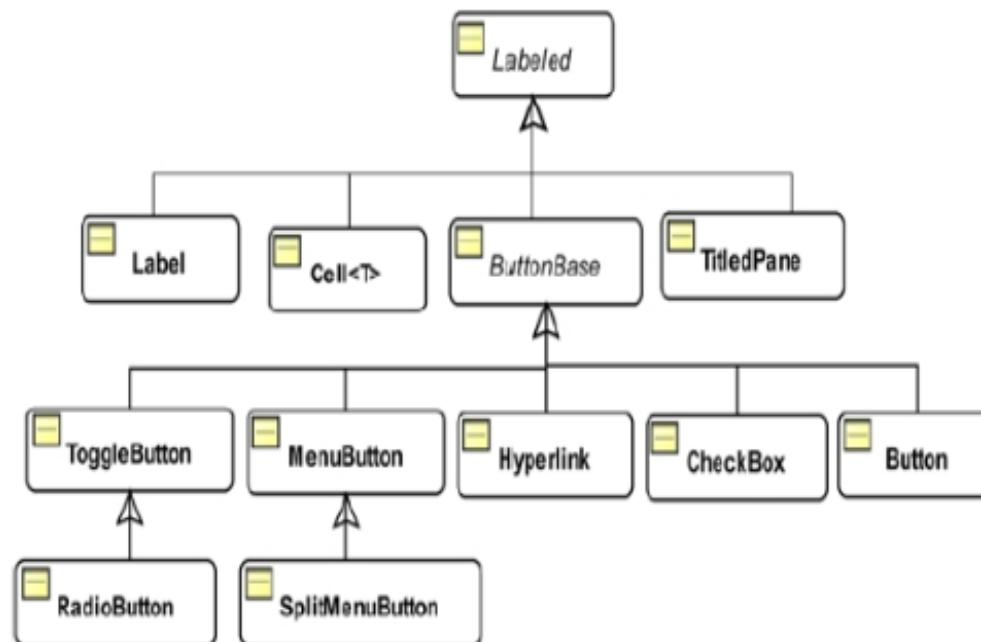
- Les composants ont tous pour classe parente **Control** qui est une sous-classe de **Node**. Une version simplifiée des dépendances entre les différentes classes est illustrée par le diagramme suivant :



## Les contrôles étiquetés (1)



- ❑ Un contrôle étiqueté contient un contenu textuel en lecture seule. Étiquette, Button, CheckBox, RadioButton et Hyperlink sont des exemples de contrôles étiquetés dans JavaFX.
- ❑ Les comportements communs de ces composants sont gérés par la classe parente **Labeled**.



## Les contrôles étiquetés (2)



- ❑ Les textes de ces composants peuvent être accompagnés d'un autre composant, généralement un graphique, une image ou une icône.
- ❑ Quelques propriétés communes aux composants **Labeled**.

text	Texte affiché ( <a href="#">String</a> ).
font	Police de caractères (famille, style, taille, ...), type <a href="#">Font</a> .
textFill	Couleur du texte, uniforme ou avec gradient (type <a href="#">Paint</a> ).
underline	Indique si le texte doit être souligné (type <a href="#">Boolean</a> ).
alignment	Alignment général du texte (et du graphique éventuel) dans la zone (type <a href="#">Pos</a> ). Valable seulement si texte sur une seule ligne.
wrapText	Booléen qui définit si le texte passe à la ligne suivante lorsqu'il atteint la limite de la zone. Le caractère ' <a href="#">\n</a> ' peut également être inséré pour forcer un retour à la ligne (inconditionnel).
textAlignment	Alignment des lignes si le texte est multiligne. Type énuméré <a href="#">TextAlignment</a> ( <a href="#">LEFT</a> , <a href="#">RIGHT</a> , <a href="#">CENTER</a> , <a href="#">JUSTIFY</a> ).
lineSpacing	Espacement des lignes pour les textes multilignes. Type <a href="#">Double</a> .

# Les contrôles étiquetés (3)

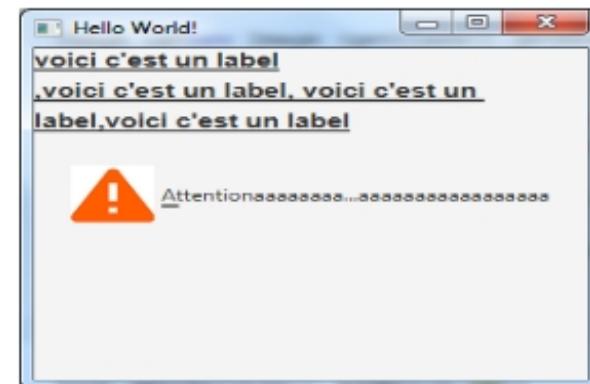


graphic	Autre composant (type <code>Node</code> ) qui accompagne le texte. Généralement un graphique, une image ou une icône.
contentDisplay	Position du composant additionnel ( <code>graphic</code> ) par rapport au texte. Type énuméré <code>ContentDisplay</code> ( <code>LEFT</code> , <code>RIGHT</code> , <code>TOP</code> , <code>BOTTOM</code> , <code>TEXT_ONLY</code> , <code>GRAPHIC_ONLY</code> ).
graphicTextGap	Espacement entre le texte et le composant additionnel ( <code>graphic</code> ). Type <code>Double</code> .
mnemonicParsing	Active le <i>parsing</i> des mnémoniques dans le texte (le caractère qui suit le caractère '_'). Type <code>Boolean</code> .
textOverrun	Comportement si le texte est trop long pour être affiché. Type énuméré <code>OverrunStyle</code> ( <code>ELLIPSIS</code> , <code>CLIP</code> , ...).
labelPadding	Définit l'espace autour du texte (et du graphique éventuel). Type <code>Insets</code> .
EllipsisString	Chaîne de caractères utilisée lorsque le texte est tronqué ( <i>ellipsis</i> ). Par défaut : "..."

\* property : Cette couleur est utilisée pour les propriétés en lecture seule (read-only)

```
Image im=new Image (getClass().getResourceAsStream("im1_1.png"));
ImageView iv=new ImageView(im);

//Label 12 =new Label ("Attention", iv);
Label 12=new Label("Attentionaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
12.setGraphic(iv);
12.setContentDisplay(ContentDisplay.LEFT);
12.setMnemonicParsing(true);
12.setTextOverrun(OverrunStyle.CENTER_WORD_ELLIPSIS);
12.setPadding (new Insets(20));
//12.setEllipsisString("----");
```





- Label :
  - ✓ Le composant Label représente un libellé (= un texte non éditable).
  - ✓ Les constructeurs permettent de définir le contenu du texte et de l'éventuel composant additionnel (graphic).
    - new Label("Hello");
    - new Label("Warning", warningIcon);
- L'essentiel des fonctionnalités sont héritées de [Labeled](#). Une seule propriété additionnelle se trouve dans [Label](#) :
  - [setLabelFor](#) : Permet de définir un composant auquel le libellé est associé



## Les contrôles étiquetés (5):Button



- Le composant Button représente un bouton permettant à l'utilisateur de déclencher une action.
- La classe parente ButtonBase rassemble les propriétés communes à différents composants qui se comportent comme des boutons : [Button](#), [CheckBox](#), [Hyperlink](#), [MenuButton](#), [ToggleButton](#)
- Les constructeurs permettent de définir le contenu du texte et de l'éventuel composant additionnel (graphic).
  - `new Button ("Ok");`
  - `new Button("Save", savelcon);`

## Les contrôles étiquetés (6):Button



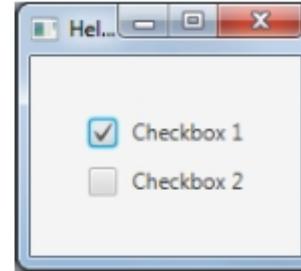
```
public void start(Stage primaryStage) {  
    Button btn = new Button();  
    VBox root = new VBox(10);  
    root.setAlignment(Pos.CENTER);  
    root.setPadding(new Insets(20));  
  
    Button btOk=new Button("OK");  
  
    Image im=new Image(this.getClass().getResourceAsStream("im2.jpg"));  
    ImageView icone=new ImageView(im);  
    Button btLog=new Button("Login",icone);  
    btLog.setContentDisplay(ContentDisplay.TOP);  
    btLog.setTextFill(Color.BLUE);  
    btLog.setGraphicTextGap(10);  
    btLog.setFont(Font.font(null, FontWeight.BOLD, 15));  
    Button btsave=new Button("Save");  
    root.getChildren().addAll(btOk,btLog,btsave);  
}
```



## Les contrôles étiquetés (7)



- CheckBox



<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/CheckBox.html>

- Hyperlink

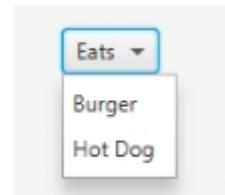
<http://example.com> — unvisited link

<http://example.com> — link is clicked

<http://example.com> — visited link

<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Hyperlink.html>

- MenuButton

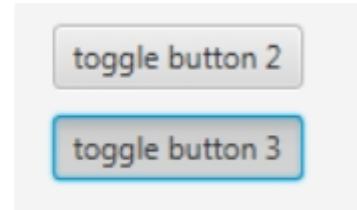


<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/MenuButton.html>

## Les contrôles étiquetés (8)



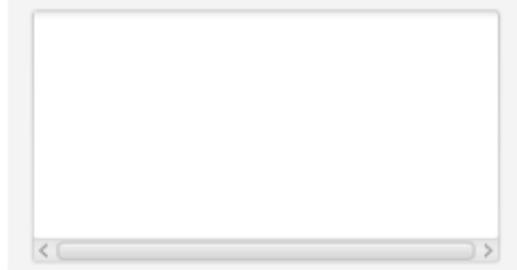
- **ToggleButton**



<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ToggleButton.html>

## Saisie de textes (1)



- La classe abstraite `TextInputControl` est la classe parente de différents composants qui permettent à l'utilisateur de saisir des textes.
- Il s'agit notamment des composants d'interface :
  - ✓ `TextField`, 
  - ✓ `PasswordField` 
  - ✓ `TextArea` 

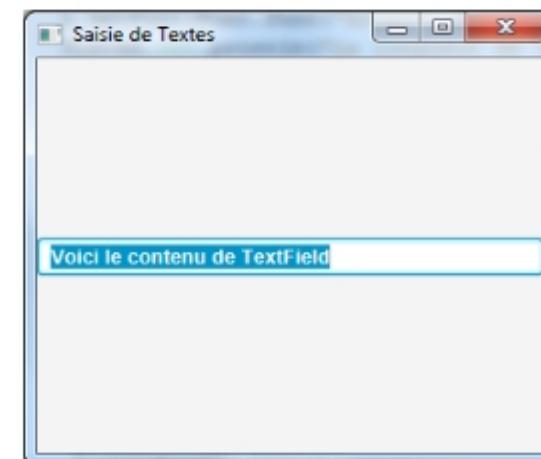
## Saisie de textes (2)



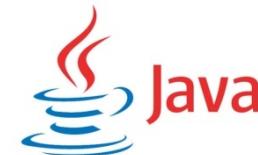
- Voilà quelques propriétés de la classe `TextInputControl`

<code>text</code>	Le texte contenu dans le composant ( <code>String</code> ).
<code>editable</code>	Le texte peut être édité par l'utilisateur ( <code>Boolean</code> ).
<code>font</code>	Police de caractères du texte ( <code>Font</code> ).
<code>length</code>	Longueur du texte ( <code>Integer</code> ).
<code>caretPosition</code>	Position courante du curseur / point d'insertion ( <code>caret</code> ).
<code>promptText</code>	Texte affiché si aucun texte n'a été défini ou saisi par l'utilisateur ( <code>String</code> ). Ce texte n'est pas affiché lorsque le composant possède le focus (avec le curseur qui clignote dans le champ). Ce texte peut éventuellement remplacer un libellé ou une bulle d'aide pour le champ texte.
<code>selectedText</code>	Texte sélectionné ( <code>String</code> ).
<code>selection</code>	Indices (de...à) de la zone sélectionnée ( <code>IndexRange</code> ).
<code>anchor</code>	Point d'ancre (début) de la sélection ( <code>Integer</code> ).

```
TextField t = new TextField();
t.setText("Voici le contenu de TextField");
t.setEditable(false);
t.setFont(Font.font("Arial", FontWeight.BOLD, 12));
System.out.println("La longueur de TextField est : "+t.getLength());
System.out.println("La position courante du curseur: "+t.getCaretPosition());
t.setPromptText("ToTo TiTi");
System.out.println("Le text sélectionné: "+t.getSelectedText());
//t.getSelectedText();
```



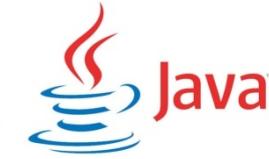
## Saisie de textes (3)



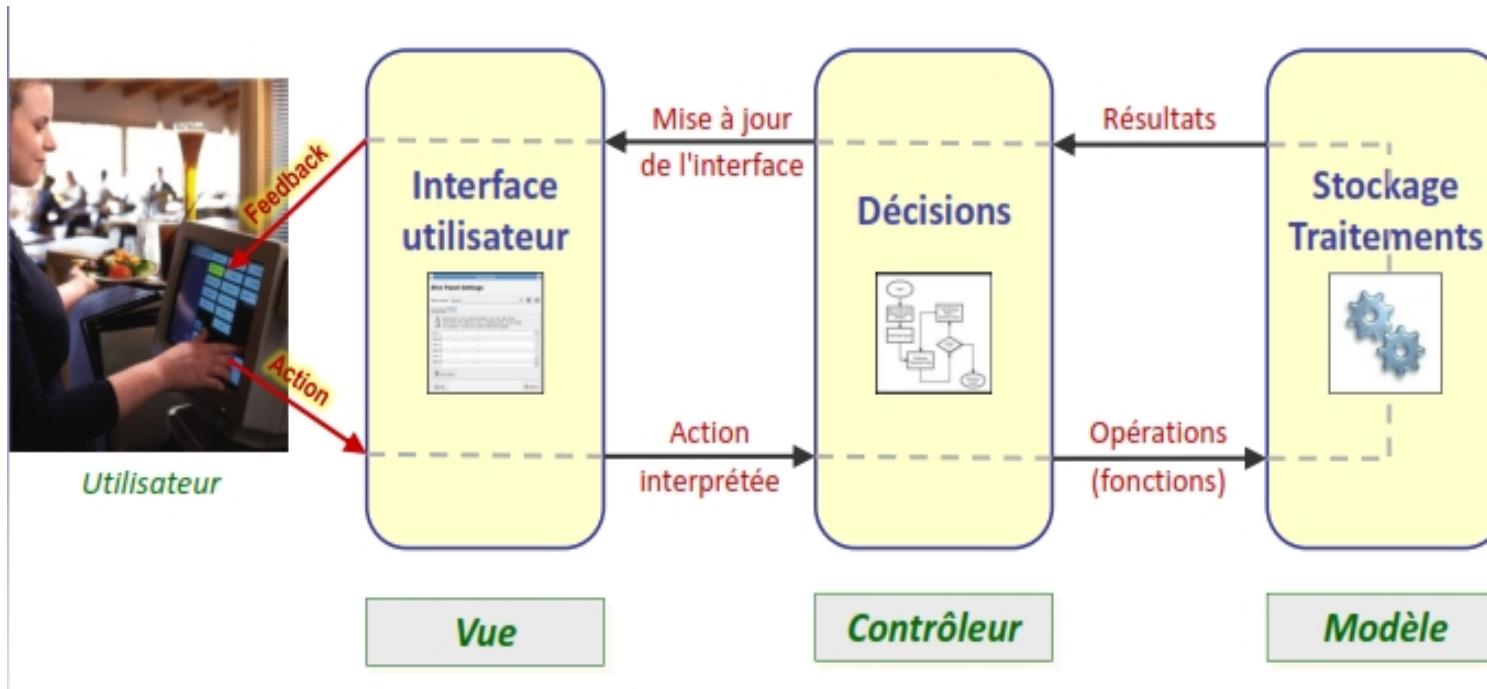
<code>clear()</code>	Efface le texte (vide le champ).
<code>copy/cut/paste()</code>	Transfert du texte dans ou depuis le <i>clipboard</i> .
<code>positionCaret()</code>	Positionne le curseur à une position donnée.
<code>forward()</code> <code>backward()</code>	Déplace d'un caractère le curseur ( <i>caret</i> ).
<code>nextWord()</code>	Déplace le curseur ( <i>caret</i> ) au début du prochain mot.
<code>insertText()</code>	Insère une chaîne de caractères dans le texte.
<code>appendText()</code>	Ajoute une chaîne de caractères à la fin du texte.
<code>deleteText()</code>	Supprime une partie du texte (de...à).
<code>deleteNextChar()</code>	Efface le prochain caractère.
<code>replaceText()</code>	Remplace une partie du texte par un autre.
<code>selectAll()</code>	Sélectionne l'ensemble du texte.
<code>deselect()</code>	Annule la sélection courante du texte.

# Gestion d'événements

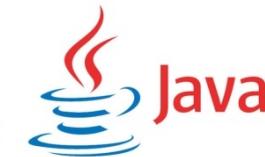
# Interactions MVC



- Lorsqu'un utilisateur interagit avec une interface, les différents éléments de l'architecture MVC interviennent pour interpréter et traiter l'événement.



# Événement (1)



- ❑ Un événement(event) constitue une notification qui signale que quelque chose s'est passé (un fait, un acte digne d'intérêt).
- ❑ Un événement peut être provoqué par :
  - ❑ **Une action de l'utilisateur**
    - ✓ Un clic avec la souris
    - ✓ La pression sur une touche du clavier
    - ✓ Le déplacement d'une fenêtre
    - ✓ Un geste sur un écran tactile... .
  - ❑ **Un changement provoqué par le système**
    - ✓ Une valeur a changé (propriété)
    - ✓ Un timer est arrivé à échéance
    - ✓ Un processus a terminé un calcul
    - ✓ Une information est arrivée par le réseau

## Événement (2)



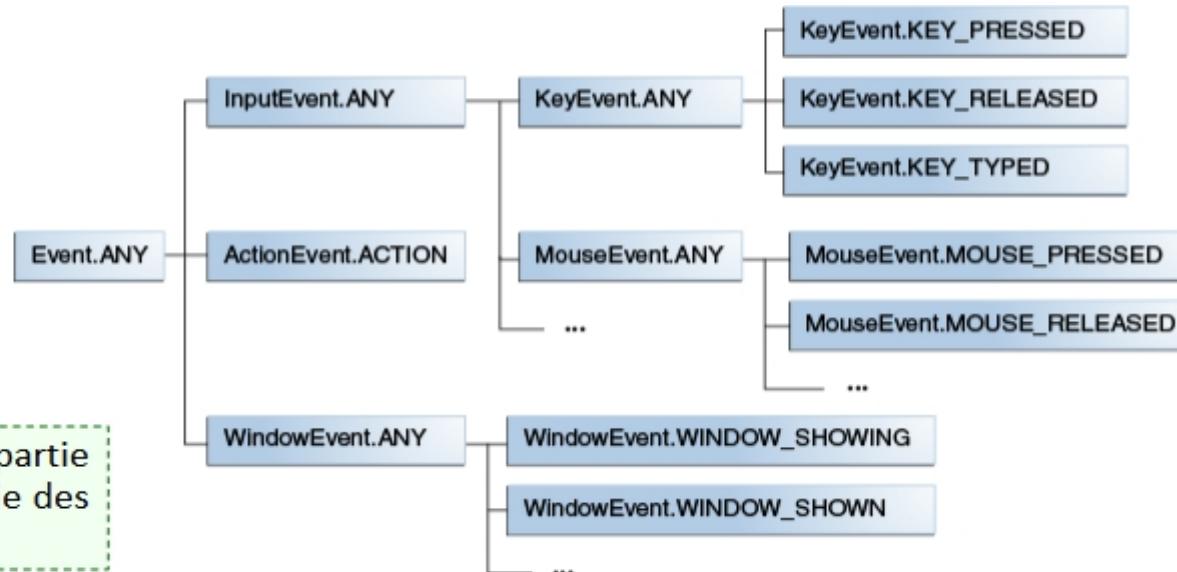
- ❑ En JavaFX les événements sont représentés par des objets de la classe **Event** ou, plus généralement, d'une de ses sous-classes.
- ❑ De nombreux événements sont prédéfinis (**MouseEvent**, **KeyEvent**, **DragEvent**, **ScrollEvent**, ...) mais il est également possible de créer ses propres événements en créant des sous-classes de **Event**.
- ❑ Chaque objet de type "événement" comprend (au moins) les informations suivantes :
  - ❑ **Le type de l'événement**(**EventTypeconsultable** avec **getEventType()**)
    - ✓ Le type permet de classifier les événements à l'intérieur d'une même classe (par exemple, la classe **KeyEvent** englobe **KEY\_PRESSED**, **KEY\_RELEASED**, **KEY\_TYPED**)
  - ❑ **La source de l'événement**(**Objectconsultable** avec **getSource()**)
    - ✓ Objet qui est à l'origine de l'événement selon la position dans la chaîne de traitement des événements (**eventdispatchchain**).
  - ❑ **La cible de l'événement**(**EventTargetconsultable** avec **getTarget()**)
    - ✓ Composant cible de l'événement (indépendamment de la position dans la chaîne de traitement des événements (**eventdispatchchain**))

# Types d'événements



- ❑ Chaque événement est d'un certain type (objet de type `EventType`).
- ❑ Chaque type d'événement possède un nom (`getName()`) et un type parent (`getSuperType()`).
- ❑ Les types d'événement forment donc une hiérarchie.
  - ❑ Par exemple si on presse une touche le nom de l'événement est `KEY_PRESSED` et le type parent est `KeyEvent.ANY`.
- ❑ A la racine, on a

`Event.ANY(=EventType.ROOT)`



# Gestion des événements(1)



□ Le traitement des événements implique les étapes suivantes :

□ **La sélection de la cible(Target) de l'événement**

- ✓ Événement clavier => le composant qui possède le focus
- ✓ Événement souris=>le composant sur lequel se trouve le curseur
- ✓ Gestes continus=>le composant au centre de la position initiale
- ✓ Si plusieurs composants se trouvent à un emplacement donné c'est celui qui est "au-dessus" qui est considéré comme la cible.

□ **La détermination de la chaîne de traitement des événements(EventDispatchChain: chemin des événements dans le graphe de scène)**

- ✓ Le chemin part de la racine (Stage) et va jusqu'au composant cible en parcourant tous les nœuds intermédiaires

□ **Le traitement des filtres d'événement(EventFilter)**

- ✓ Exécute le code des filtres en suivant le chemin **descendant**, de la racine (Stage) jusqu'au composant cible

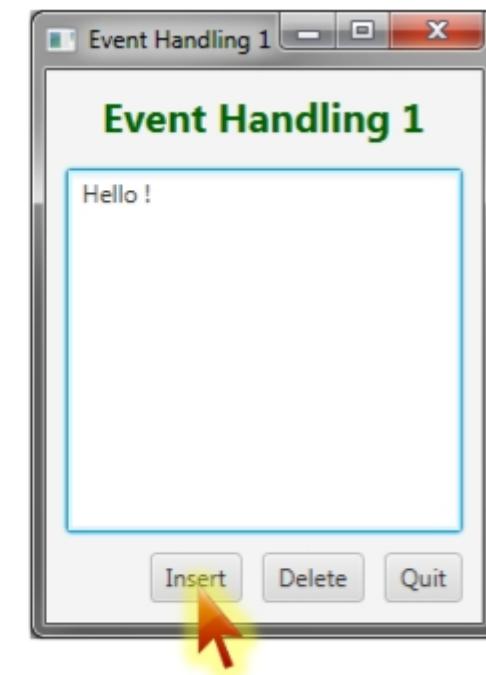
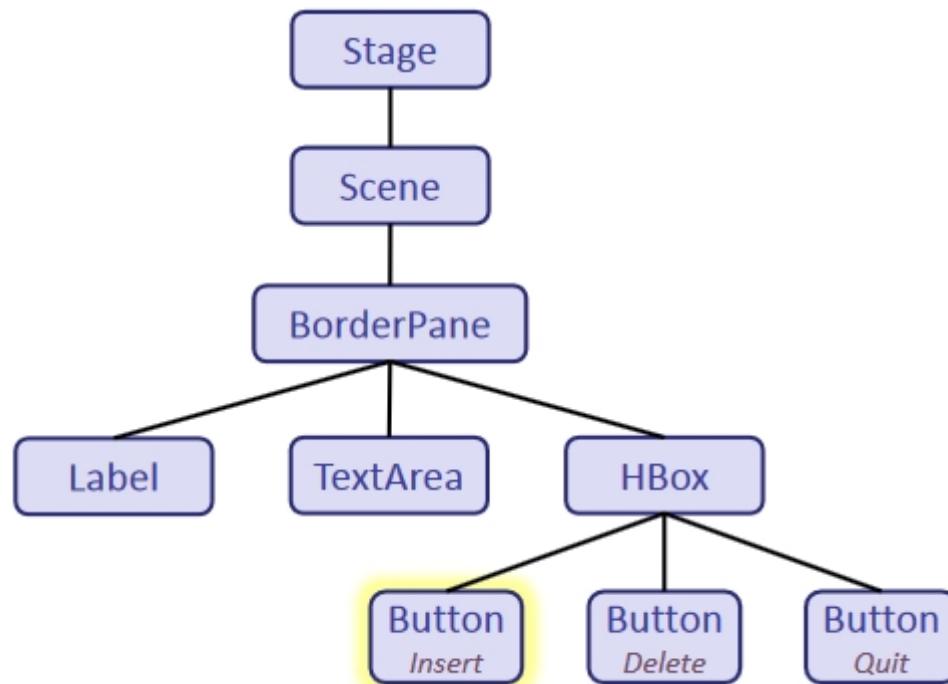
□ **Le traitement des gestionnaires d'événement(EventHandler)**

- ✓ Exécute le code des gestionnaires d'événement en suivant le chemin **montant**, du composant cible à la racine (Stage)

## Gestion des événements(2)



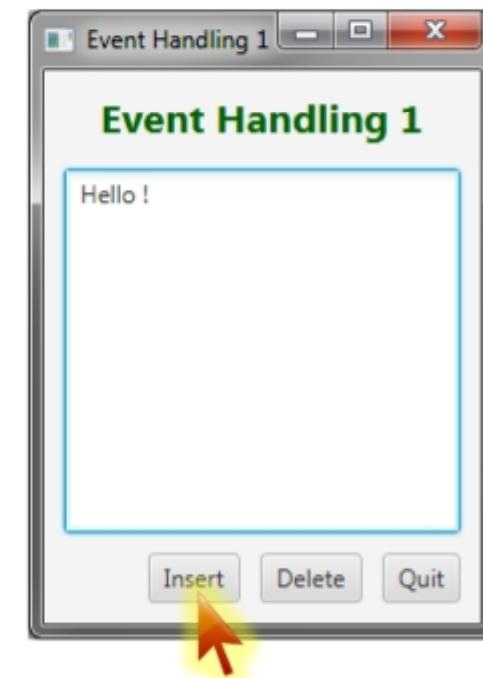
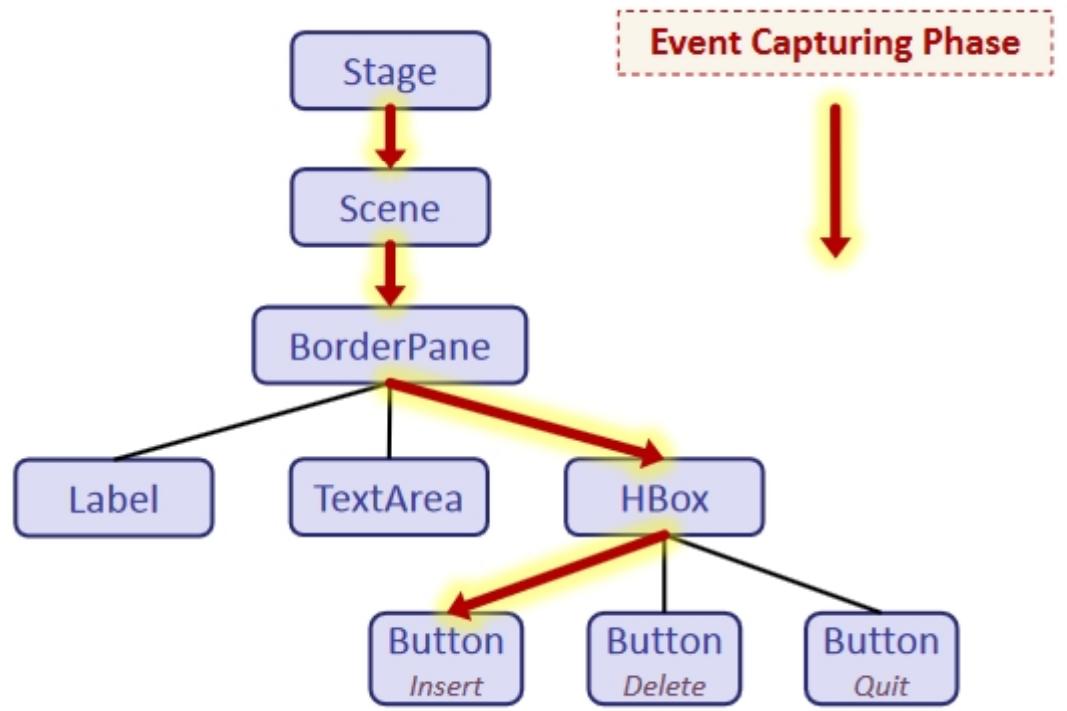
- Un exemple d'application avec son graphe de scène.
- Si l'utilisateur clique sur le bouton **Insert**, un événement de type **Action** va être déclenché et va se propager le long du chemin correspondant à la chaîne de traitement (**EventDispatchChain**).



# Gestion des événements(3)



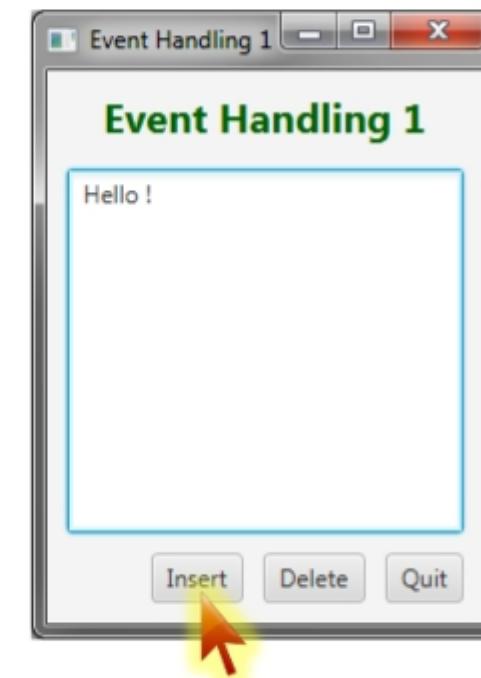
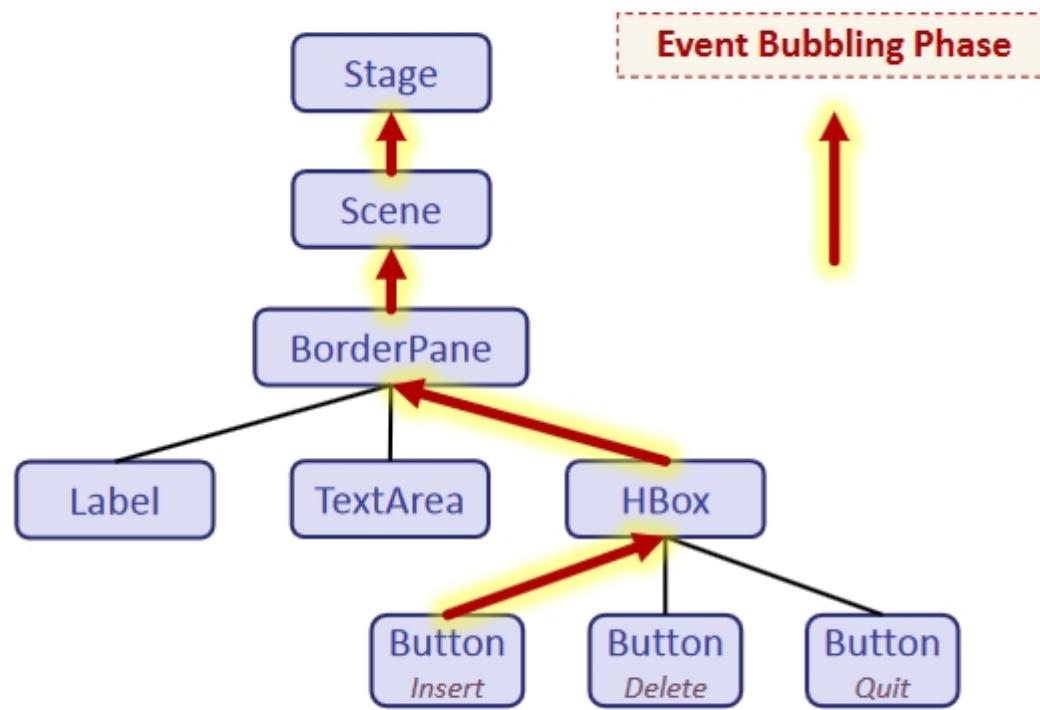
- L'événement se propage d'abord vers le bas, depuis le nœud racine (Stage) jusqu'à la cible (Target)- c'est-à-dire le bouton cliqué -et les filtres(EventFilter) éventuellement enregistrés sont exécutés(dans l'ordre de passage).



# Gestion des événements(4)



- L'événement remonte ensuite depuis la cible jusqu'à la racine et les gestionnaires d'événements(**EventListener**)éventuellement enregistrés sont exécutés(dans l'ordre de passage).



# Gestion des événements(5)

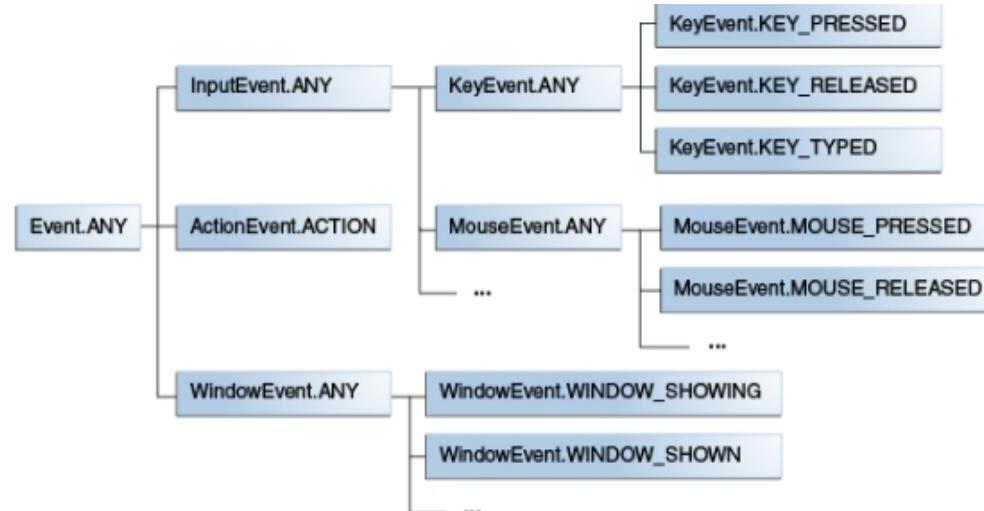


- ❑ Pour gérer un événement (exécuter des instructions), il faut créer un récepteur d'événement(**EventListener**), appelé aussi **écouteur d'événement**, et l'enregistrer sur les nœuds du graphe de scène où l'on souhaite intercepter l'événement et effectuer un traitement.
- ❑ Un récepteur d'événement peut être enregistré comme filtre ou comme gestionnaire d'événement. La différence principale entre les deux réside dans le moment où le code est exécuté :
  - ❑ Les **filtres(filters)**sont exécutés dans la phase **descendante** de la chaîne de traitement des événements (avant les gestionnaires)
  - ❑ Les **gestionnaires(handlers)**sont exécutés dans la phase **montante** de la chaîne de traitement des événements (après les filtres)
- ❑ Les filtres, comme les gestionnaires d'événements, sont des objets qui doivent implémenter l'interface fonctionnelle (générique) **EventHandler<T extends Event>**qui impose l'unique méthode **handle(T event)**qui se charge de traiter l'événement.

# Gestion des événements(8)



- ❑ Si un nœud du graphe de scène possède plusieurs récepteurs d'événements enregistrés, l'ordre d'activation de ces récepteurs sera basé sur la hiérarchie des types d'événement :
  - ❑ Un récepteur pour un type spécifique sera toujours exécuté avant un récepteur pour un type plus générique
  - ❑ Par exemple un filtre enregistré pour `MouseEvent.MOUSE_PRESSED` sera exécuté avant un filtre pour `MouseEvent.ANY` qui sera exécuté avant un filtre pour `InputEvent.ANY`
  - ❑ L'ordre d'exécution des récepteurs pour des types de même niveau n'est pas défini
  - ❑ La consommation d'un événement n'interrompt pas le traitement des autres récepteurs enregistrés sur le même nœud



# Event Handling (1)

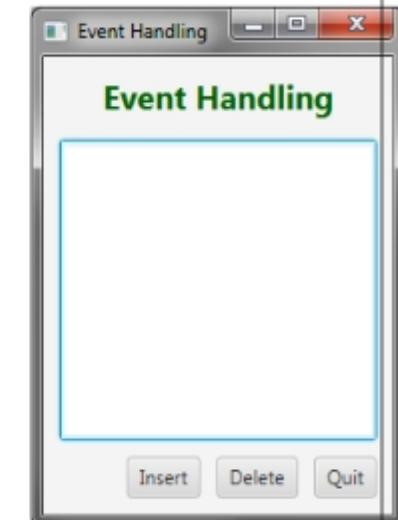


```
private BorderPane root      = new BorderPane();
private HBox      btnPanel  = new HBox(10);
private Label    lblTitle   = new Label("Event Handling");
private TextArea txaMsg     = new TextArea();
private Button   btnInsert  = new Button("Insert");
private Button   btnDelete  = new Button("Delete");
private Button   btnQuit    = new Button("Quit");
```

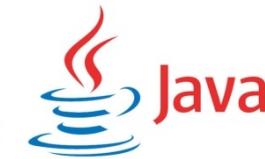
```
primaryStage.setTitle("Event Handling");
root.setPadding(new Insets(10));

//--- Title
lblTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
lblTitle.setTextFill(Color.DARKGREEN);
BorderPane.setAlignment(lblTitle, Pos.CENTER);
BorderPane.setMargin(lblTitle, new Insets(0, 0, 10, 0));
root.setTop(lblTitle);

//--- Text-Area
txaMsg.setWrapText(true);
txaMsg.setPrefColumnCount(15);
txaMsg.setPrefRowCount(10);
root.setCenter(txaMsg);
```

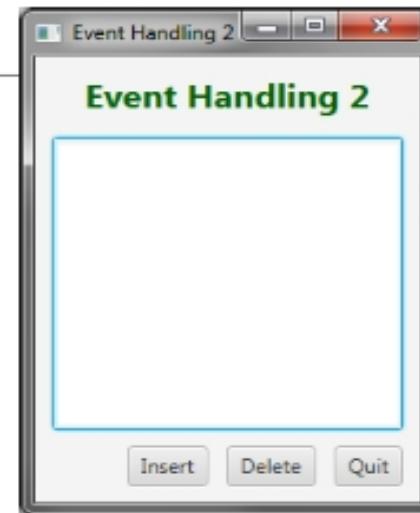


# Event Handling (2)

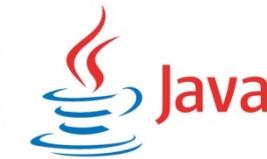


```
//--- Button Panel  
btnPanel.getChildren().add(btnInsert);  
btnPanel.getChildren().add(btnDelete);  
btnPanel.getChildren().add(btnQuit);  
btnPanel.setAlignment(Pos.CENTER_RIGHT);  
btnPanel.setPadding(new Insets(10, 0, 0, 0));  
root.setBottom(btnPanel);  
  
Scene scene = new Scene(root);  
primaryStage.setScene(scene);  
primaryStage.show();
```

Code de l'interface **sans** la gestion des événements.

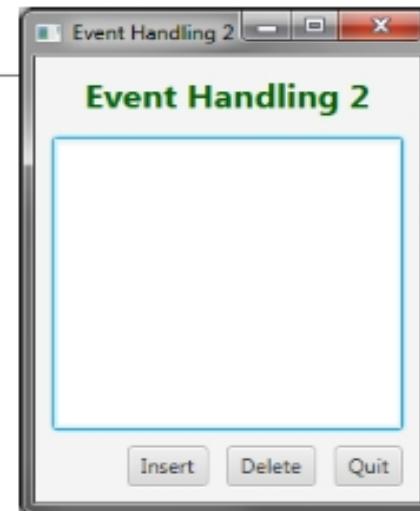


## Event Handling (2)



```
//--- Button Panel  
btnPanel.getChildren().add(btnInsert);  
btnPanel.getChildren().add(btnDelete);  
btnPanel.getChildren().add(btnQuit);  
btnPanel.setAlignment(Pos.CENTER_RIGHT);  
btnPanel.setPadding(new Insets(10, 0, 0, 0));  
root.setBottom(btnPanel);  
  
Scene scene = new Scene(root);  
primaryStage.setScene(scene);  
primaryStage.show();
```

Code de l'interface **sans** la gestion des événements.



# Event Handling (3)



- Pour traiter les événements des boutons, on peut créer **une classe contrôleur** qui implémente **EventHandler** et effectue les opérations souhaitées dans la méthode **handle()**.

```
public class InsertButtonController implements EventHandler<ActionEvent> {  
    private TextArea tArea;  
    //--- Constructeur -----  
    public InsertButtonController(TextArea tArea) {  
        this.tArea = tArea;  
    }  
    //--- Code exécuté lorsque l'événement survient -----  
    @Override  
    public void handle(ActionEvent event) {  
        tArea.appendText("A");  
    }  
}
```

*Si on veut agir sur des composants de la vue,  
il faut transmettre les références nécessaires.*

*Si le code est plus complexe,  
on invoquera de préférence  
une méthode de la vue.*

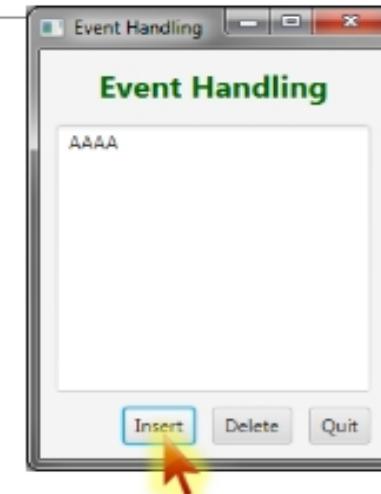
## Event Handling (4)



- Dans la vue, il faut ensuite créer une instance de ce contrôleur et l'enregistrer comme gestionnaire d'événement (**type ACTION**) sur le bouton concerné en invoquant la méthode **addEventHandler()**.

```
 * * *
//--- Button Events Handling
InsertButtonController insertCtrl = new InsertButtonController(txMsg);
btnInsert.addEventHandler(ActionEvent.ACTION, insertCtrl);
* * *
```

- A chaque clic sur le bouton *Insert*, le gestionnaire d'événement sera exécuté et un caractère '*A*' sera ajouté dans le composant *TextArea*.



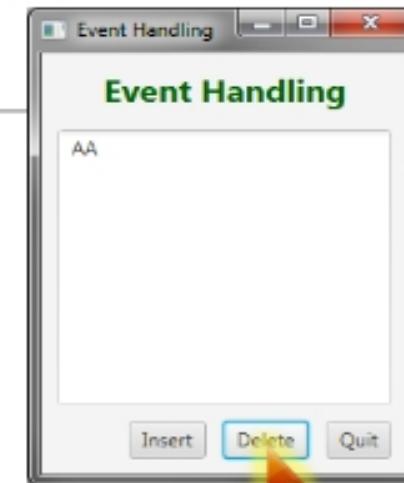
## Event Handling (5)



- Une autre manière de faire consiste à créer le contrôleur sous la forme d'une classe locale anonyme. Par exemple, pour le bouton Delete:

```
    ...
    //--- Button Events Handling
    btnDelete.addEventHandler(ActionEvent.ACTION,
        new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                txaMsg.deletePreviousChar();
            }
        });
    ...
    ...
```

- A chaque clic sur le bouton *Delete*, le gestionnaire d'événement sera exécuté et un caractère sera supprimé dans le composant **TextArea**.



## Event Handling (6)



- Une troisième possibilité pour traiter les événements des boutons, est d'utiliser la méthode **setOnAction()** et passer en paramètre une expression lambda implémentant la méthode **handle()** de l'interface **EventHandler**.
- Par exemple pour traiter les trois boutons de l'interface :

```
//--- Button Events Handling
btnInsert.setOnAction(event -> {
    txaMsg.appendText("A");
});

btnDelete.setOnAction(event -> {
    txaMsg.deletePreviousChar();
});

btnQuit.setOnAction(event -> {
    Platform.exit();
});
```

# Classe contrôleur



- Dans la variante MVC synchrone, il est fréquent que la classe du contrôleur reçoive dans son constructeur les références du modèle et de la vue.)

```
public class ButtonController implements EventHandler<ActionEvent> {  
    private AppModel model;  
    private MainView view;  
  
    //--- Constructeur -----  
    public ButtonController(AppModel model, MainView view) {  
        this.model = model;  
        this.view = view;  
    }  
  
    //--- Code exécuté lorsque l'événement survient -----  
    @Override  
    public void handle(ActionEvent event) {  
        int newVal = model.getInfo();  
        view.updateInfo(newVal);  
    }  
}
```

*Références du modèle et de la vue.*

*Le contrôleur accède aux données du modèle et met à jour la vue.*

# Méthodes setOn...(1)



- Liste des principales actions associées à des méthodes utilitaires qui permettent d'enregistrer des gestionnaires d'événements (il faut rechercher les méthodes **setOnEventType() dans la classe**).

Action de l'utilisateur	Événement	Dans classe
Pression sur une touche du clavier	KeyEvent	Node, Scene
Déplacement de la souris ou pression sur une de ses touches	MouseEvent	Node, Scene
Glisser-déposer avec la souris ( <i>Drag-and-Drop</i> )	MouseDragEvent	Node, Scene
Glisser-déposer propre à la plateforme (geste par exemple)	DragEvent	Node, Scene
Composant "scrollé"	ScrollEvent	Node, Scene
Geste de rotation	RotateEvent	Node, Scene
Geste de balayage/défilement ( <i>swipe</i> )	SwipeEvent	Node, Scene
Un composant est touché	TouchEvent	Node, Scene
Geste de zoom	ZoomEvent	Node, Scene
Activation du menu contextuel	ContextMenuEvent	Node, Scene

# Méthodes setOn...(2)



Action de l'utilisateur	Événement	Dans classe
Texte modifié (durant la saisie)	InputMethodEvent	Node, Scene
Bouton cliqué ComboBox ouverte ou fermée Une des options d'un menu contextuel activée Option de menu activée Pression sur <i>Enter</i> dans un champ texte	ActionEvent	ButtonBase ComboBoxBase ContextMenu MenuItem TextField
Élément ( <i>Item</i> ) d'une liste, ... d'une table ou ... d'un arbre a été édité	ListView. EditEvent TableColumn. CellEditEvent TreeView. EditEvent	ListView TableColumn TreeView
Erreur survenue dans le <i>media-player</i>	MediaErrorEvent	MediaView
Menu est affiché (déroulé) ou masqué (enroulé)	Event	Menu
Fenêtre <i>popup</i> masquée	Event	PopupWindow
Onglet sélectionné ou fermé	Event	Tab
Fenêtre affichée, fermée, masquée	WindowEvent	Window

# Documentation officielle



- ❑ Javadoc (à partir de celle de JavaSE)

<https://docs.oracle.com/javase/8/javafx/api/toc.htm>

- ❑ Documentation et exemples de JavaFX

<https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

- ❑ Tutoriel sur les composants de l'interface

<https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/>

- ❑ Tutoriel sur les layouts

<https://docs.oracle.com/javase/8/javafx/layout-tutorial/>

- ❑ Tutoriel sur la gestion des événements

<https://docs.oracle.com/javase/8/javafx/events-tutorial/>