UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

DÉVELOPPEMENT DES INTERFACES HOMME MACHINE (IHM)

Nourhène BEN RABAH

Doctorante en Automatique et Génie Informatique
Centre de Recherche en STIC (CReSTIC), URCA
https://crestic.univ-reims.fr/fr/ben-rabah

Séance 18 Octobre 2018

Nourhene.Ben-Rabah@univ-paris1.fr

Présentation

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

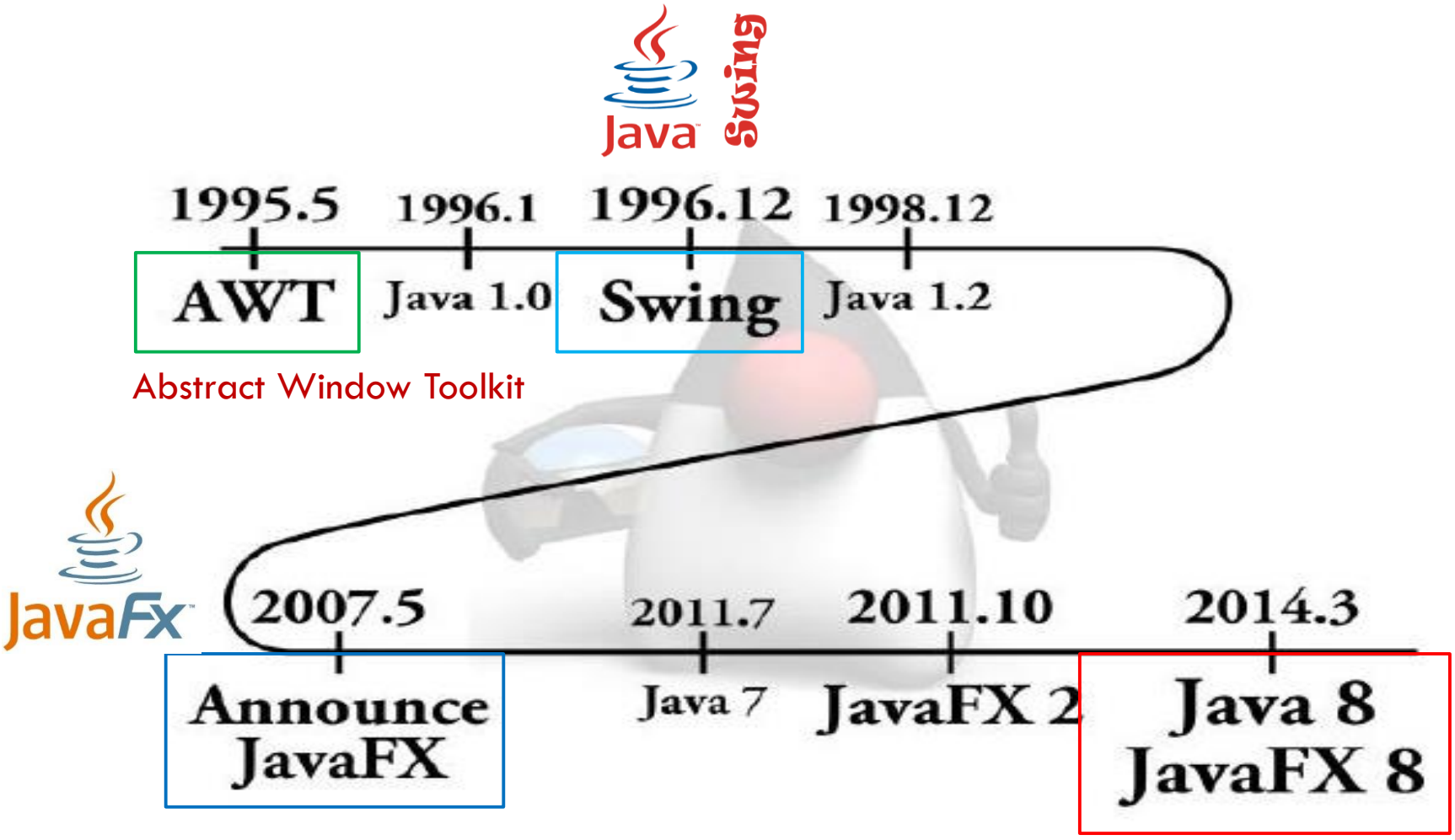
- **Objectif** :
  - Comment concevoir et réaliser des interfaces Homme-Machine en Java
- **Organisation** :
  - 4 séances de 6h
  - ½ concepts + ½ travaux pratiques
- **Évaluation :**
  - Contrôle continu : travaux en cours + projet
  - Examen

Présentation

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

- **Contenu prévisionnel**
  - Initiation à JavaFx : les conteneurs et les contrôleurs
  - JavaFx : les événements
  - Modèle MVC (Partie 1)
  - Modèle MVC (Partie 2)
  - Projet

Interfaces graphiques avec Java : Bref historique (1)



Interfaces graphiques avec Java : Bref historique (2)

- 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)*
  - Essayer de concurrencer Flex (qui se base sur Flash +MXML)

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)

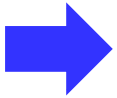
- La plateforme JavaFX offre deux techniques complémentaires pour créer les interfaces graphiques des applications :

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

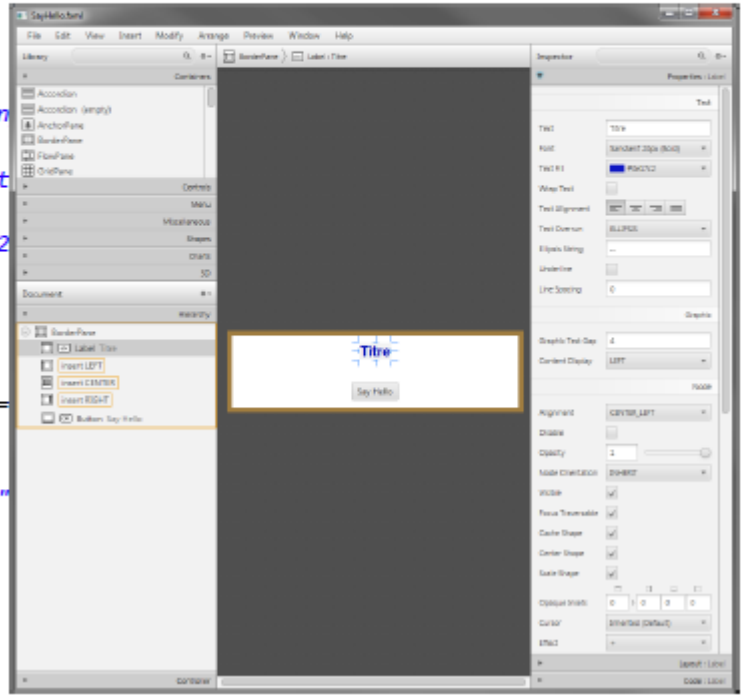


Il est possible de mélanger les deux techniques au sein d'une même application

Des interfaces déclaratives vs procédurales (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>
```



JavaFX : Configuration requise

- Pour développer des applications JavaFx, vous devez installer ces logiciels :
  - Java Development Kit 8
    http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
  - L'IDE NetBeans 8.0 ou les versions ultérieurs
    https://netbeans.org/downloads/

Manière procédurale : Concepts techniques

JavaFX : concepts de base

- Comment se présente une application *JavaFX* ?

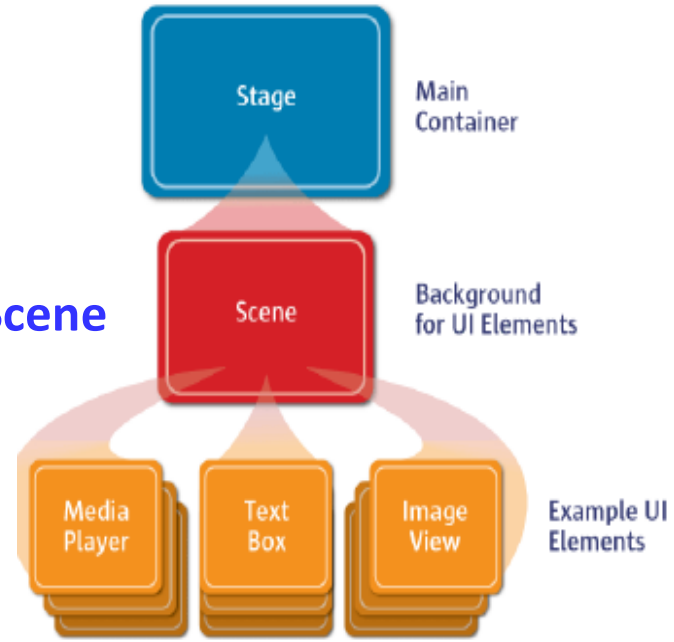
(petite immersion avant de décrire plus en détail les concepts de base).

- 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

- Les éléments structurels principaux d'une application JavaFX se basent sur la Métaphore de la salle de spectacle (theater)

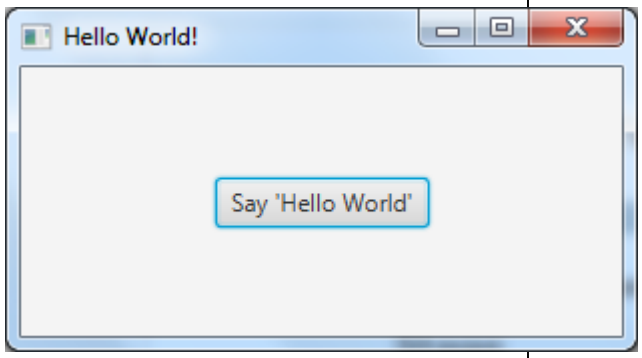
Remarque : En français, on utilise le terme 'scène' pour parler de l'endroit où se passe le spectacle (l'estrade, les planches) mais également pour parler de ce qui s'y déroule (jouer ou tourner une scène) ce qui peut conduire à un peu de confusion avec cette métaphore.

- **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)

L'application traditionnel Hello World :

```
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);
    }

}
```



JavaFX : première application (2)

- 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 :

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

JavaFX : première application (4)

Sans main ()

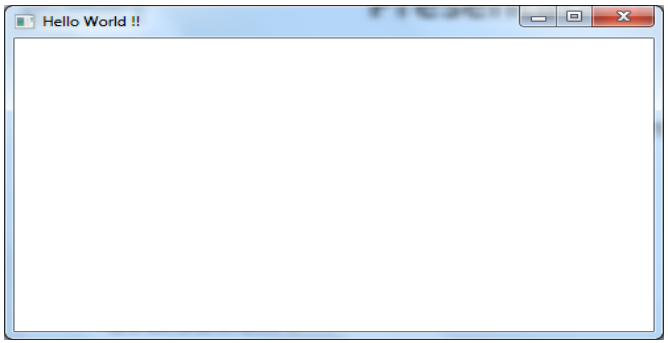
- Java qui lance l'application JavaFX si la classe en cours d'exécution ne contient pas la méthode main()

```
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 ()

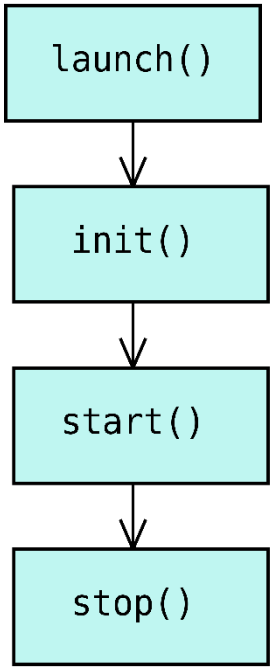
- If faut appeler la méthode launch () de la classe Application

```
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

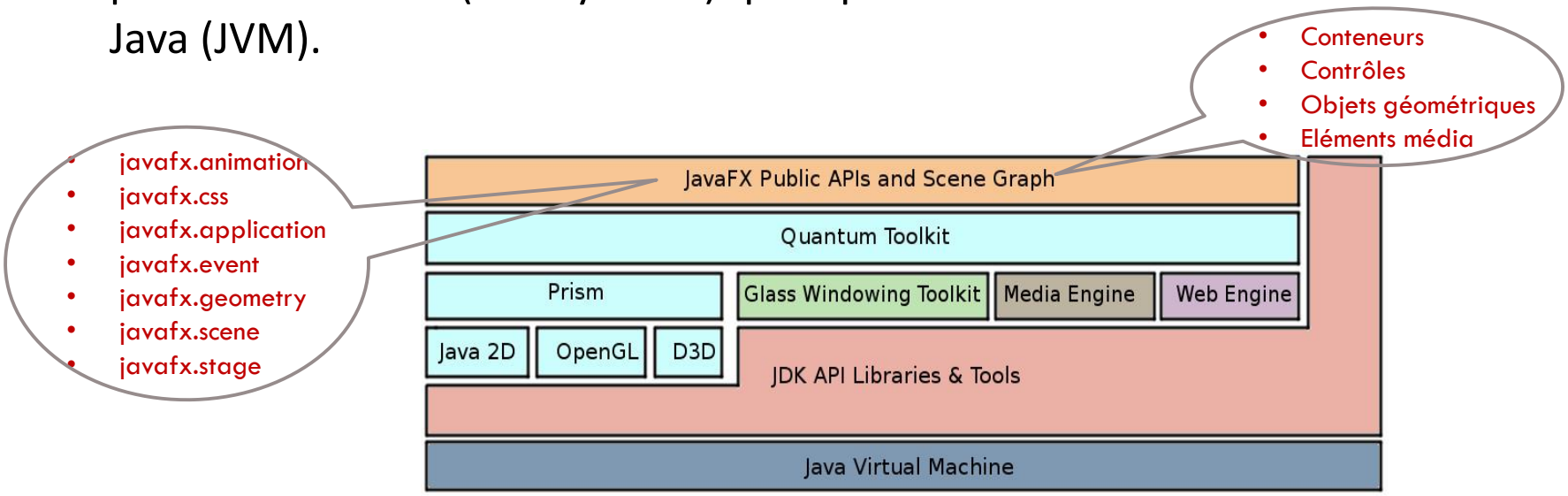
- Le point d'entrée d'une application JavaFX est constitué de l'instance de la classe Application (généralement une sous classe).
- Lors du lancement d'une application par la méthode statique **Application. Launch ()** le runtime JavaFX effectue les opérations suivantes :
  1. Crée une instance de la 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()**



TP1 Partie 1

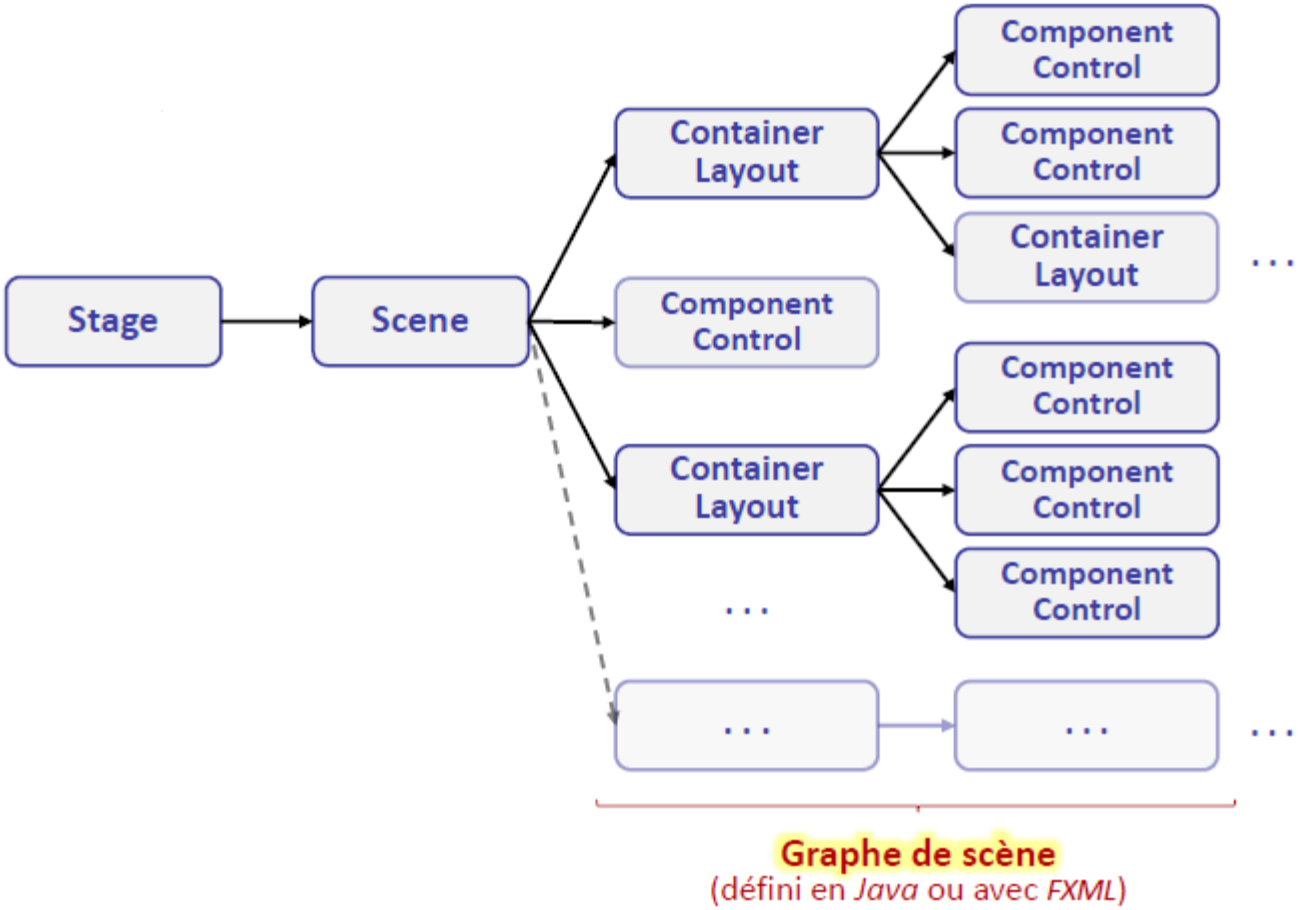
Architecture JavaFx

- 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)

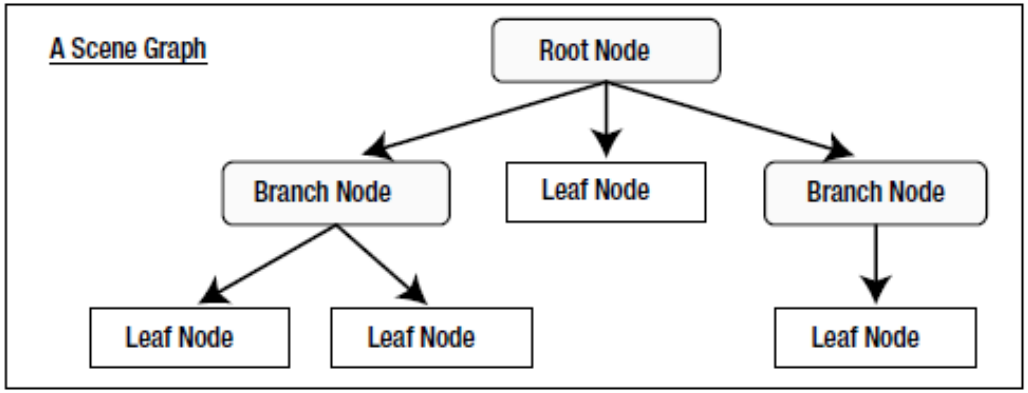
Eléments d'une application JavaFX



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

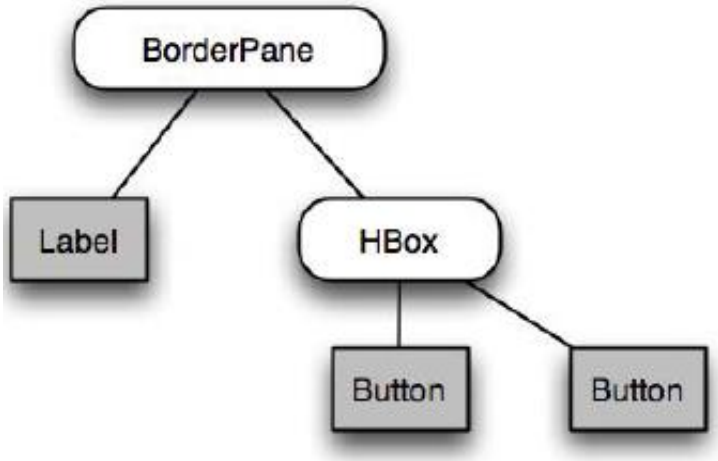
- Les noeuds (nodes) peuvent être de trois types :
  - Racine
  - Noeud intermédiaire
  - Feuille (leaf)



Graphe de scène (2)

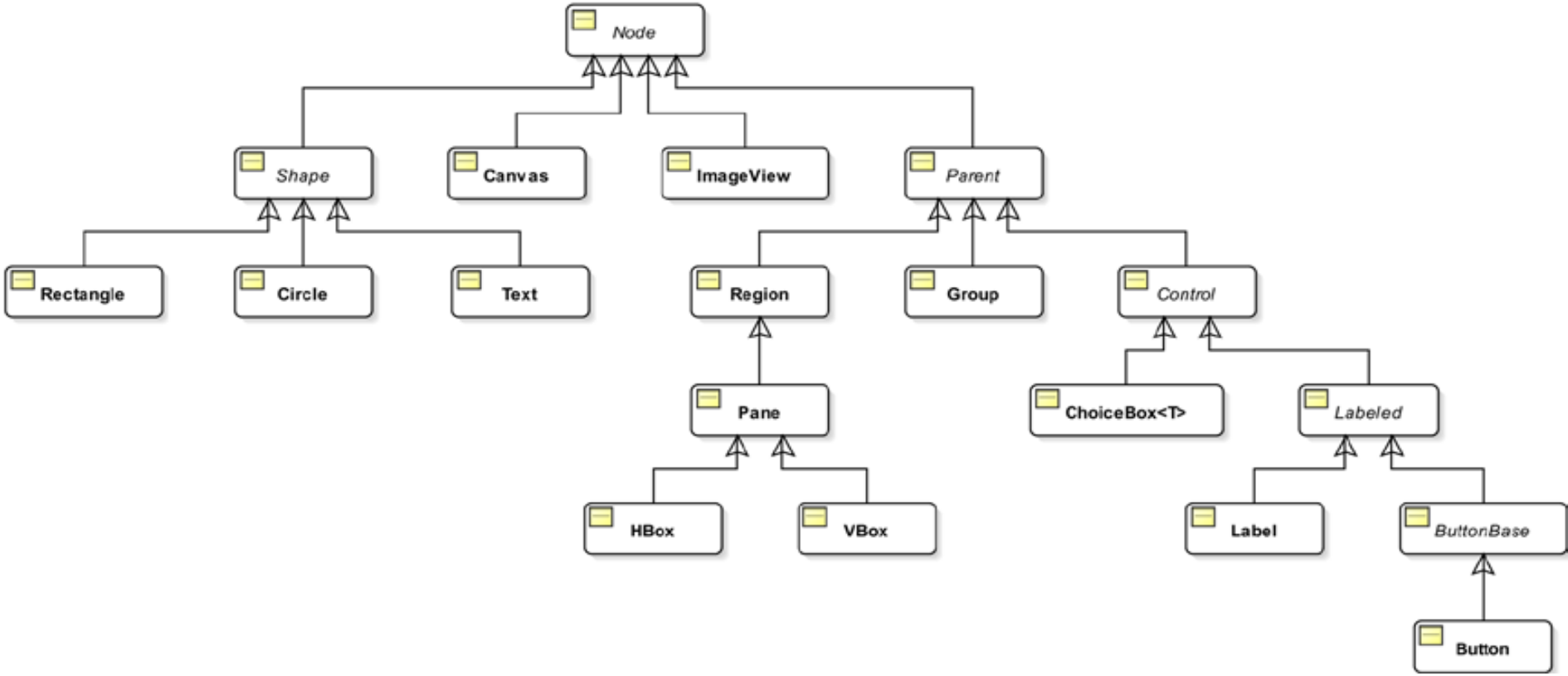
- Les feuilles de l'arbre sont généralement constitués de **composants visibles** (**boutons**, **champs texte**, …)
- les nœuds intermédiaires et le nœud racine sont généralement des éléments de structuration (souvent **invisibles),** typiquement **des conteneurs** de différents types (HBox, VBox, BorderPane,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

Exemple :



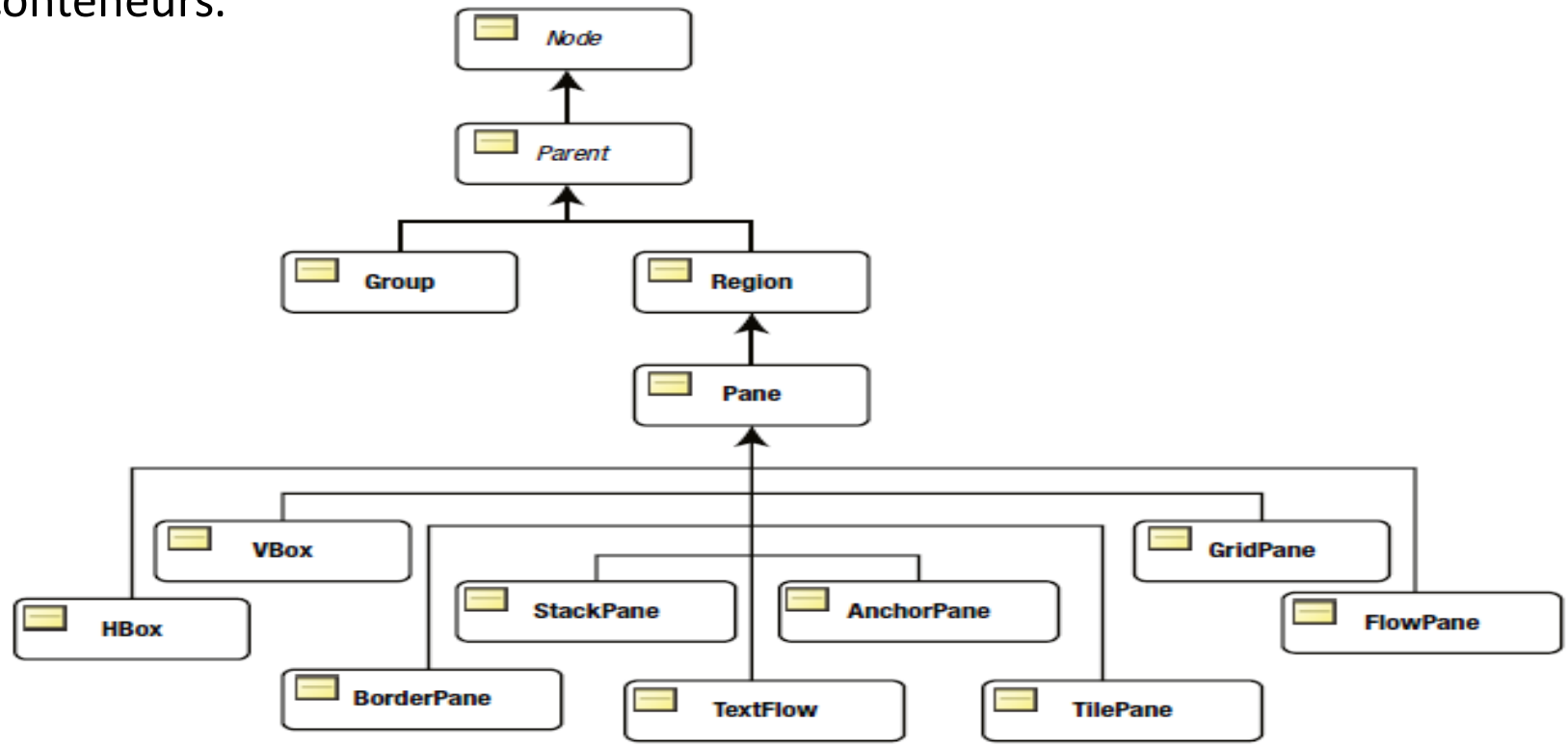
Graphe de scène (3)

- 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 :



Conteneurs

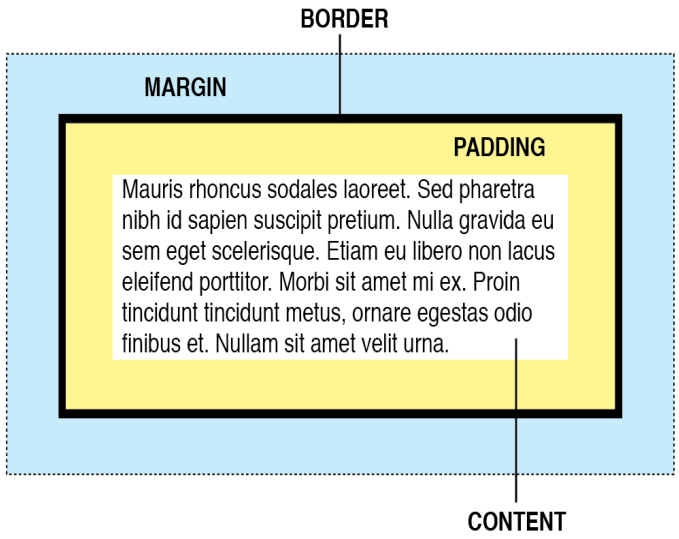
- Les **conteneurs** ***(Layout-Pane)*** représentent une famille importante parmi les sous-classes de Node. Ils ont pour classe parente **Pane** et **Region** qui possèdent de nombreuses propriétés et méthodes héritées par tous les conteneurs.



Region - Structure visuelle

- 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, …).

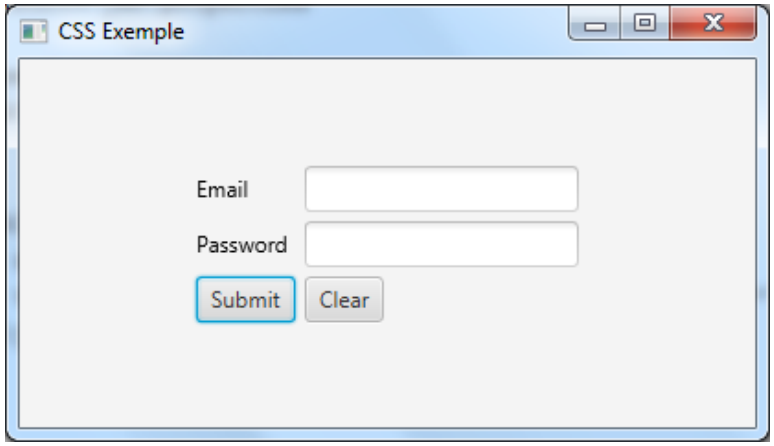
- En *JavaFX*, le style des composants est défini par des feuilles de style de type

CSS. Il est ainsi possible de changer globalement l'aspect de l'interface sans avoir à modifier le code de l'application.

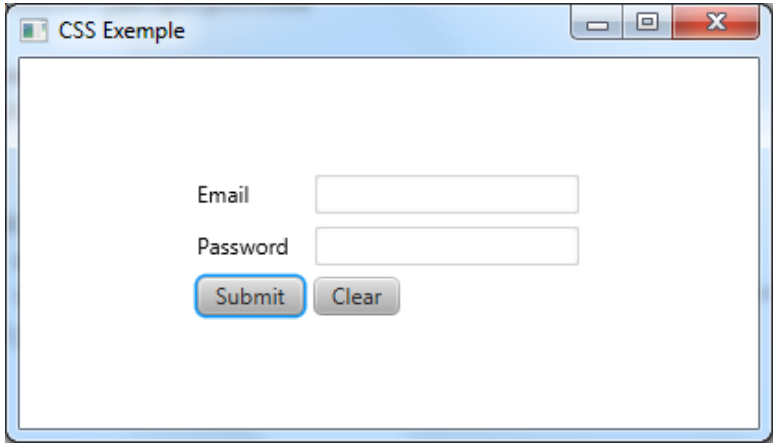
- La méthode setUserAgentStylesheet () de la classe Application permet d'indiquer l'URL de la feuille de style qui est à appliquer globalement.

- 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());

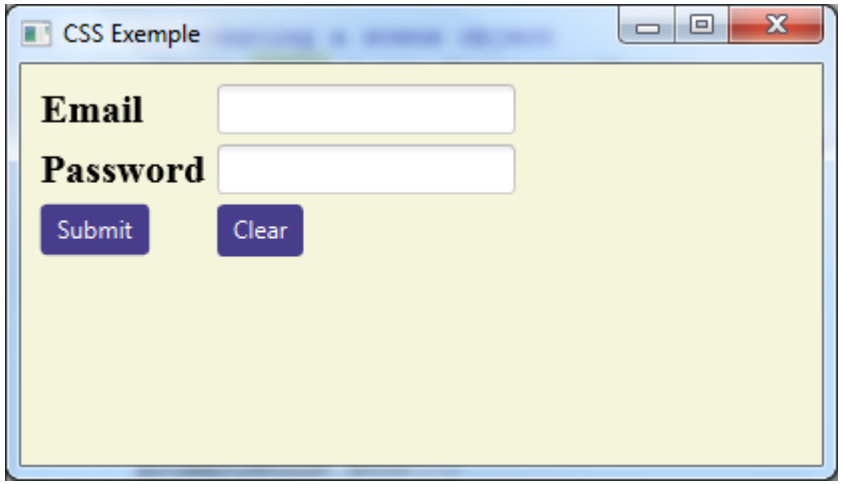
Fichier Style.css

```
.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 ;

}
GridPane{
     -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;");
```

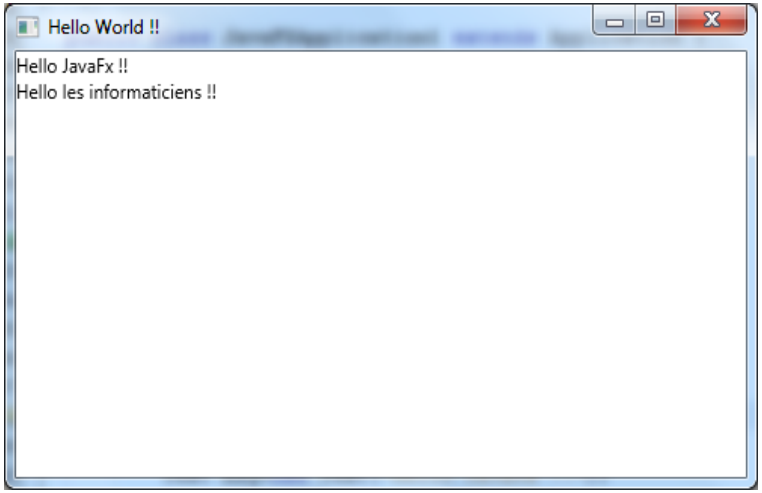


Les conteneurs

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();
     }
```

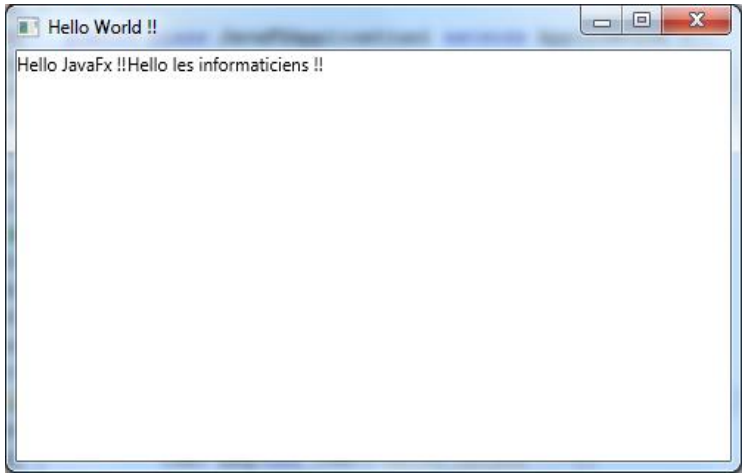
Vbox : (**V**ertical **box**): arrange les sous-composants sur une seule colonne;



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();

}
```

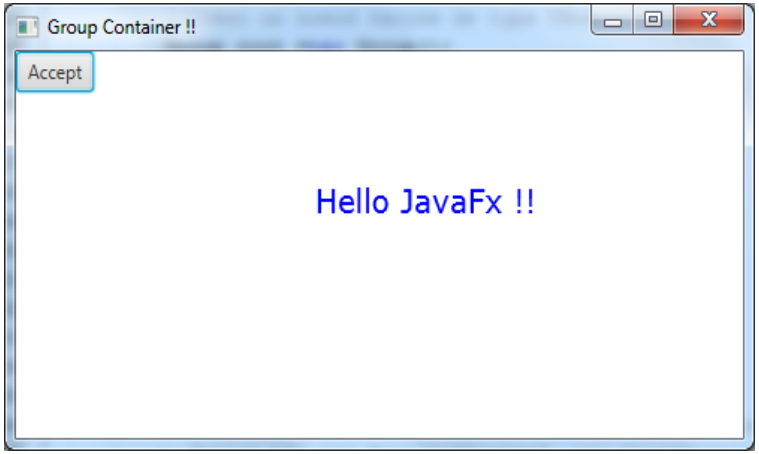
Hbox : (**H**orizontal **box**): arrange les sous-composants sur une seule ligne;



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);
```

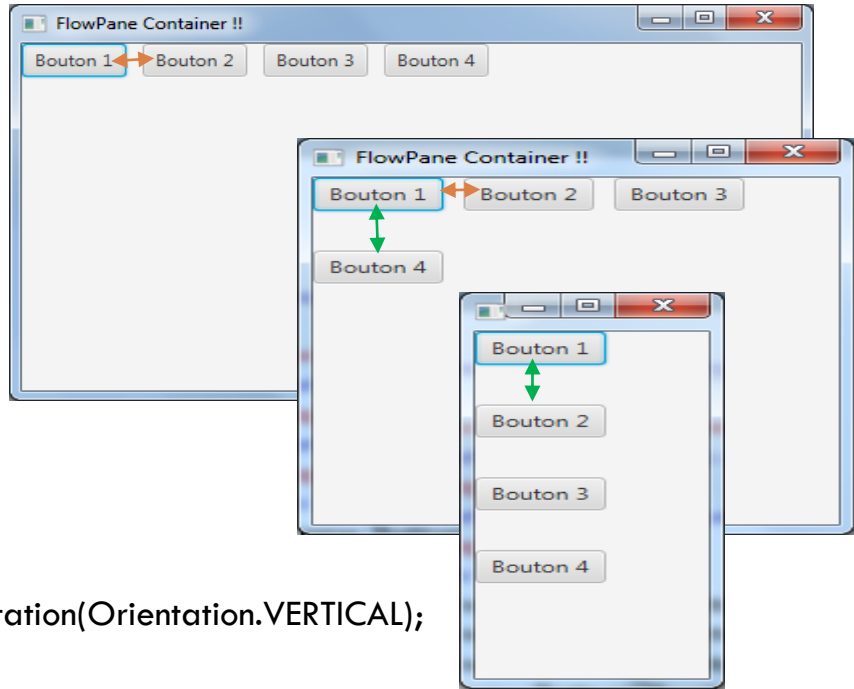
Group : n'applique pas aucune disposition pour ses sous-composants
Tous les sous-composants sont dans la position 0,0



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("Boutton1");
Button b2=new Button("Boutton2");
Button b3=new Button("Boutton3");
Button b4=new Button("Boutton4");
//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.

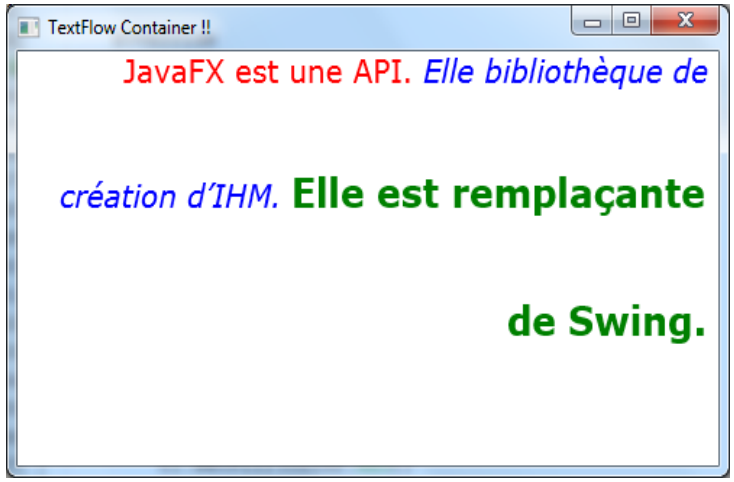


root.setOrientation(Orientation.VERTICAL);

Le conteneur Textflow


```
TextFlow root=new TextFlow();
//l'alignement des objets Text dans le conteneur
root.setTextAlignment(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);
```

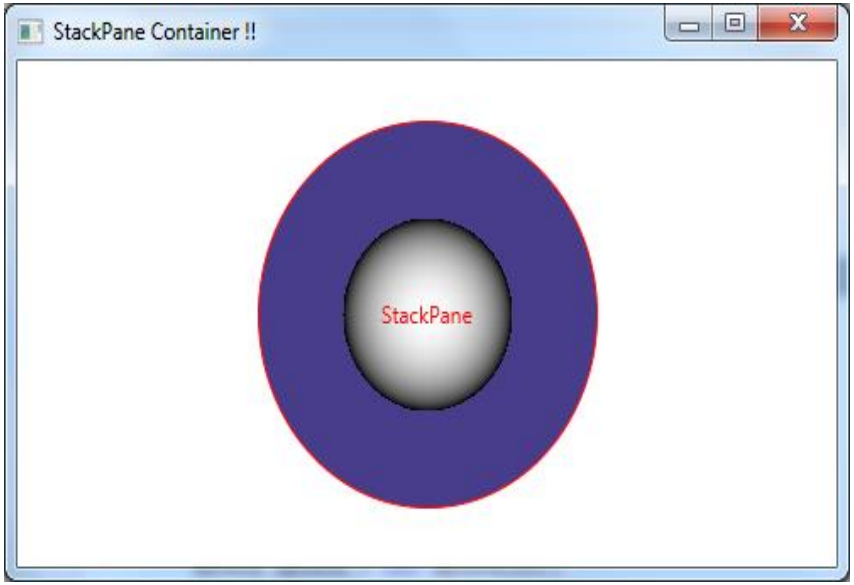
Textflow : ranger plusieurs nœuds text (CENTER, JUSTIFY, LEFT, RIGHT)



Le conteneur StackPane


```
//Dessiner un cercle
Circle circle = new Circle(300, 135, 100);
//Couleur du Remplissage
circle.setFill(Color.DARKSLATEBLUE);
//Couleur du conteur
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

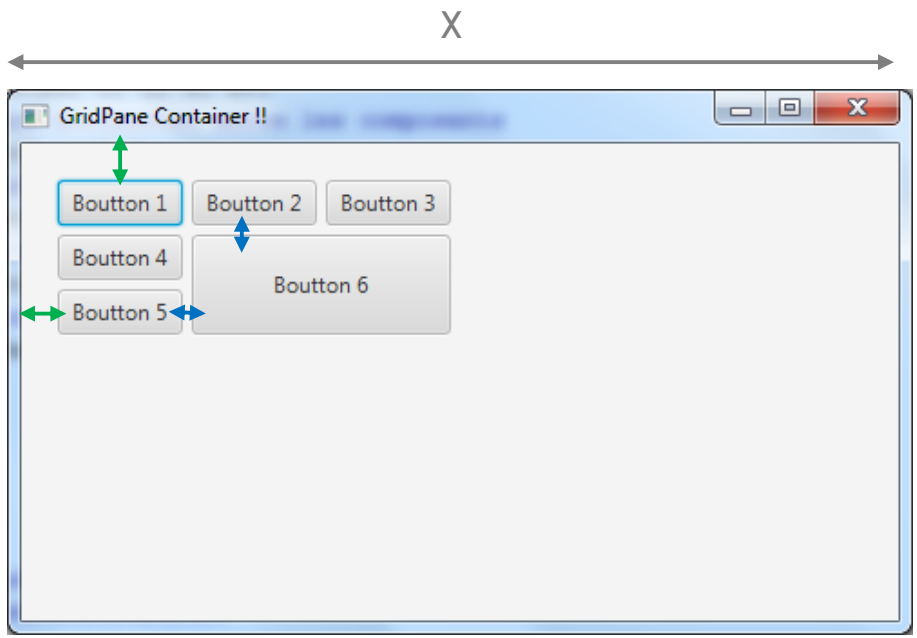


Le conteneur GridPane


```
GridPane root=new GridPane();
//Bouttons
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");
//Alignement 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);
```

TOP_RIGHT
BOTTOM_LEFT
BOTTOM_RIGHT
CENTER

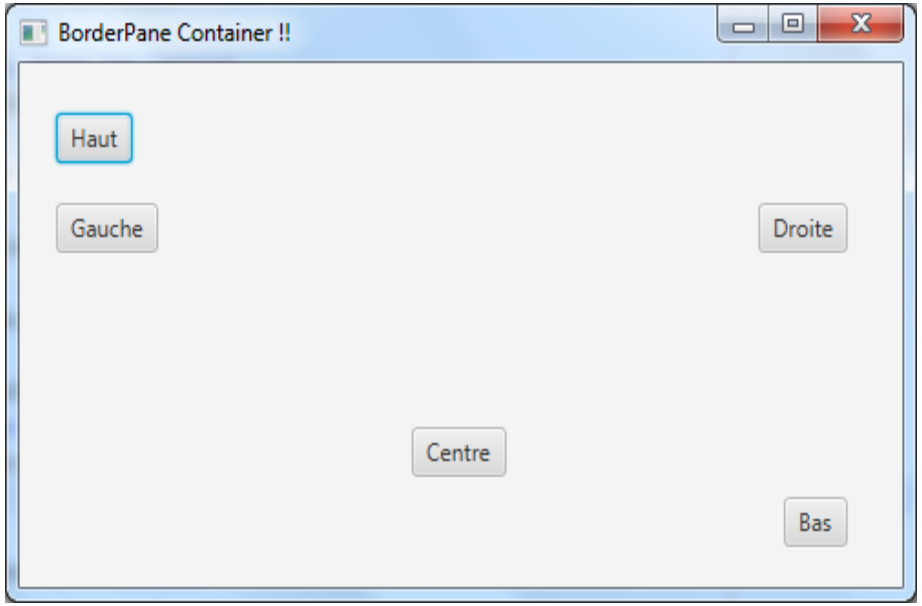
GridPane :
organiser ses sous-composants dans une grille.



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.



TP1 Partie 2

Les composants de base : création et utilisation

Les contrôles (1)

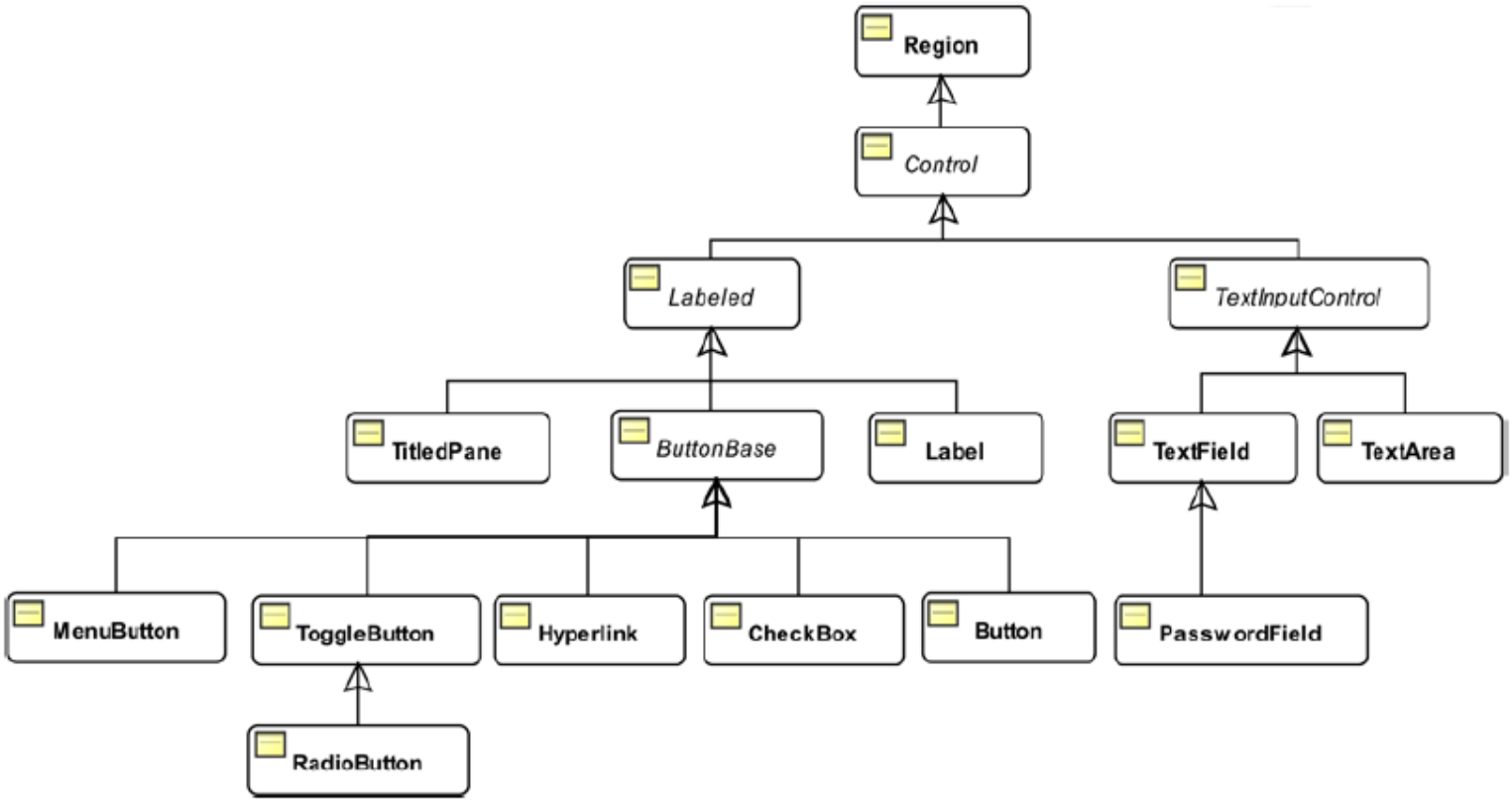
- JavaFX offre un ensemble de composants (kit de développement) pour créer les interfaces utilisateurs graphiques.
- Ces composants d'interface sont fréquemment nommés controls dans la documentation en anglais (parfois widgets).

- 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)

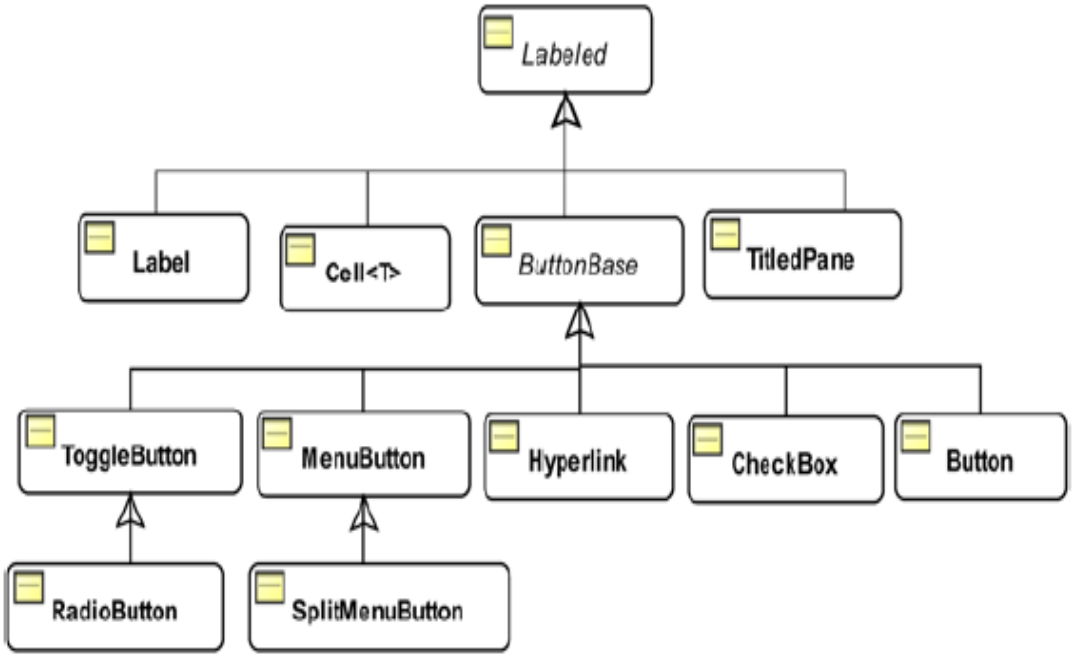
- Les composants ont tous pour classe parente **Control** qui est une sous-classe de **Node**. Une version simplifiée des dépendances entre les différentes classes est illustrée par le diagramme suivant :



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**.

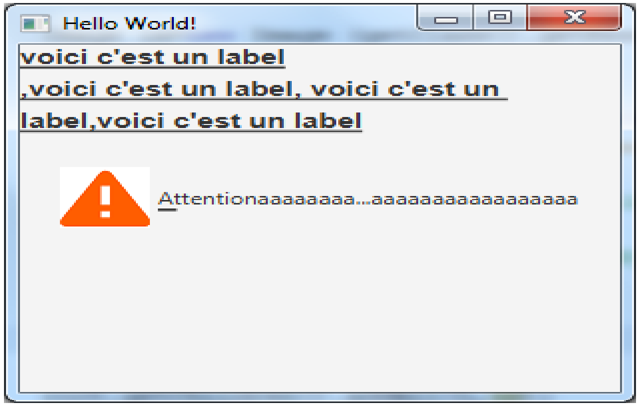
`text`	Texte affiché (`String`).
`font`	Police de caractères (famille, style, taille, ...), type `Font`.
`textFill`	Couleur du texte, uniforme ou avec gradient (type `Paint`).
`underline`	Indique si le texte doit être souligné (type `Boolean`).
`alignment`	Alignement général du texte (et du graphique éventuel) dans la zone (type `Pos`). 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 '\n' peut également être inséré pour forcer un retour à la ligne (inconditionnel).
`textAlignment`	Alignement des lignes si le texte est multiligne. Type énuméré `TextAlignment` (`LEFT`, `RIGHT`, `CENTER`, `JUSTIFY`).
`lineSpacing`	Espacement des lignes pour les textes multilignes. Type `Double`.

Les contrôles étiquetés (3)

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

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

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

//Label l2 =new Label ("Attention", iv);
Label l2=new Label("_Attentionaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
l2.setGraphic(iv);
l2.setContentDisplay(ContentDisplay.LEFT);
l2.setMnemonicParsing(true);
l2.setTextOverrun(OverrunStyle.CENTER_WORD_ELLIPSIS);
l2.setPadding (new Insets(20));
//l2.setEllipsisString("----");
```



Les contrôles étiquetés (4) : Label

- 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

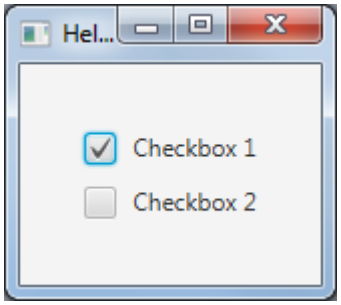
- Les constructeurs permettent de définir le contenu du texte et de l'éventuel composant additionnel (graphic).
  - new Button ("Ok");
  - new Button("Save", saveIcon);

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

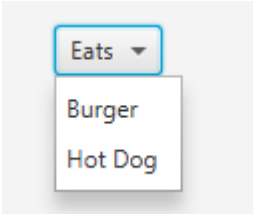


https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/CheckBox.html

- Hyperlink

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

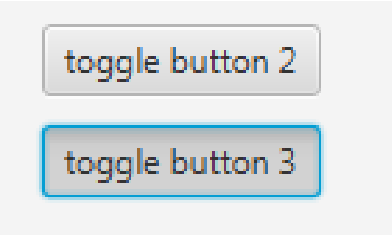
- MenuButton



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

Les contrôles étiquetés (8)

- ToggleButton

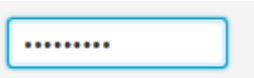


https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ToggleButton.html

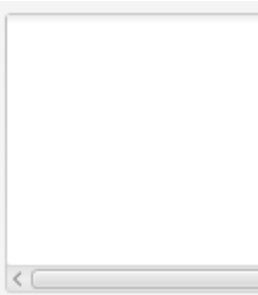
Saisie de textes (1)

- 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

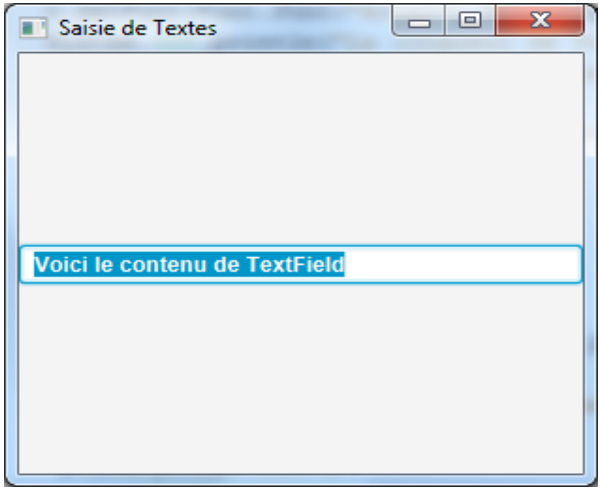


Saisie de textes (2)

- Voila quelques propriétés de la classe TextInputControl

text	Le texte contenu dans le composant (String).
editable	Le texte peut être édité par l'utilisateur (Boolean).
font	Police de caractères du texte (Font).
length	Longueur du texte (Integer).
caretPosition	Position courante du curseur / point d'insertion *(caret)*.
promptText	Texte affiché si aucun texte n'a été défini ou saisi par l'utilisateur (String). 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.
selectedText	Texte sélectionné (String).
selection	Indices (de...à) de la zone sélectionnée (IndexRange).
anchor	Point d'ancrage (début) de la sélection (Integer).

```
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)

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

TP1 Partie 3