Cours IHM-1
JavaFX

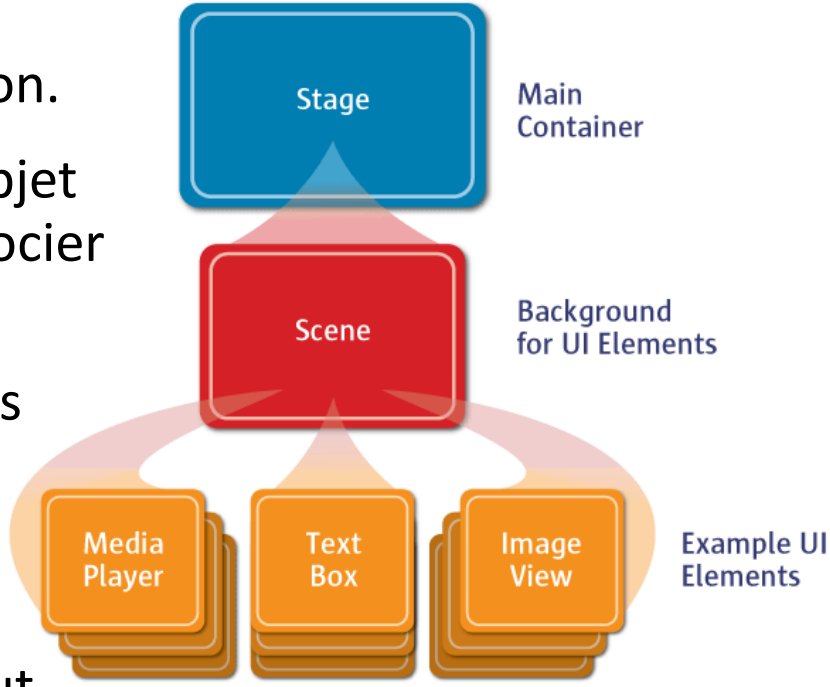
3 - Architecture
Concepts techniques

Une première immersion

(Hello World !)

Première application [1]

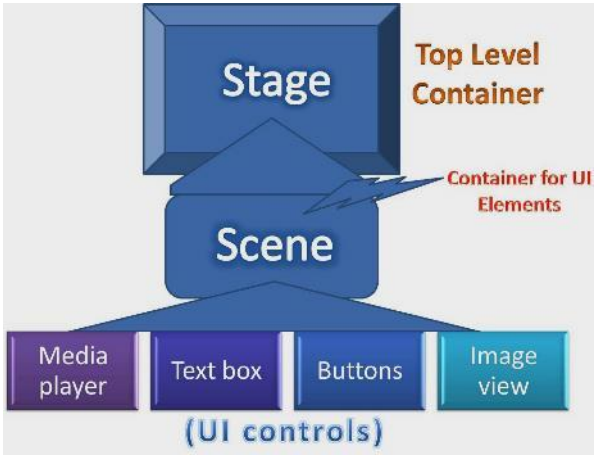
- Comment se présente une application *JavaFX* ?
  (petite immersion avant de décrire plus en détail les concepts de base).
- L'application est codée en créant une sous-classe de **`Application`**.
- La fenêtre principale d'une application est représentée par un objet de type **`Stage`** qui est fourni par le système au lancement de l'application.
- 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`**.
- La méthode **`start()`** construit le tout.



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.*



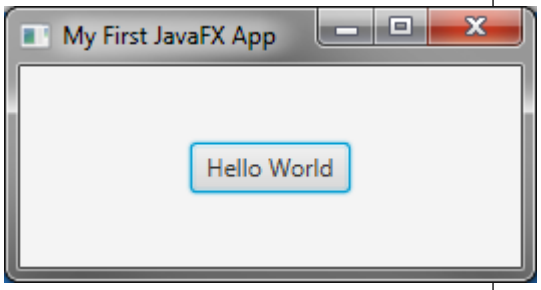
- **Stage** : L'endroit où a lieu l'action, où se déroule la scène
- **Scene** : Tableau ou séquence faisant intervenir les acteurs
- **Controls** : Acteurs, figurants, éléments du décor, … (éléments actifs/passifs) qui font partie de la scène en train d'être jouée.

  Components
  Nodes
  Widgets

Bienvenue [1]

- Une première application, le traditionnel *Hello World !*

```
public class Bienvenue  extends Application {
  //-----------------------------------------------
  @Override
  public void start(Stage premierStage) {
    premierStage.setTitle("My First JavaFX App");
    BorderPane root = new BorderPane();
    Button btnHello = new Button("Hello World");
    root.setCenter(btnHello);
    Scene scene = new Scene(root, 250, 100);
    premierStage.setScene(scene);
    premierStage.show();
  }

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



Bienvenue [2]

- *JavaFX* étant intégré à la plateforme de base *Java*, aucune librairie externe n'est nécessaire au fonctionnement de l'exemple précédent.
- Un certain nombre d'importations doivent cependant être effectuées (on y retrouve les classes principales `Application`, `Stage`, `Scene` ainsi que les composants `BorderPane` et `Button`) :

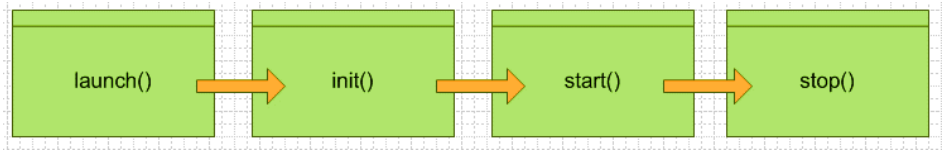
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
```

En principe, Eclipse s'en charge automatiquement

- Dans cet exemple, la méthode `main()` lance uniquement la méthode statique `launch()` (qui est héritée de `Application`).
  - Dans une application un peu plus complexe, elle pourrait effectuer d'autres opérations d'initialisation avant l'invoquer `launch()`.

Cycle de vie d'une application [1]

- 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 (et `Platform.isImplicitExit()` retourne `true`)
     - ✓ L'application appelle `Platform.exit()` (ne pas utiliser `System.Exit()`)
  5. Appelle la méthode **`stop()`**



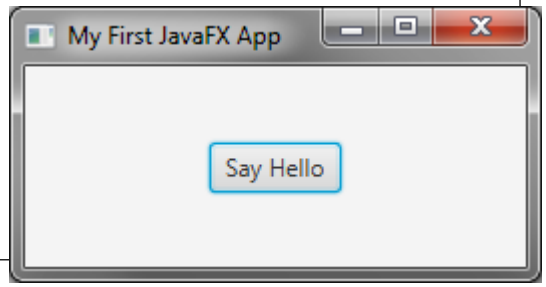
Cycle de vie d'une application [2]

- La méthode `launch()` est généralement lancée depuis la méthode `main()`. Elle est implicitement lancée s'il n'y a pas de méthode `main()` (ce qui est toléré depuis *Java* 8).
- Des paramètres de lancement peuvent être récupérés en invoquant la méthode **`getParameters()`** dans la méthode `init()` ou ultérieurement.
  - Invoquer ensuite `getRaw()` ou `get`… pour obtenir la liste (`List<String>`)
- La méthode `start()` est abstraite et doit être redéfinie.
- Les méthodes `init()` et `stop()` ne doivent pas obligatoirement être redéfinies (par défaut elle ne font rien).
- La méthode `start()` s'exécute dans le *JavaFX Application Thread*. C'est dans ce *thread* que doit être construite l'interface et que doivent être exécutées toutes les opérations qui agissent sur des composants attachés à une *scène (live components)*.

Traiter une action de l'utilisateur

- Dans l'exemple *Hello World*, pour que le clic sur le bouton déclenche une action, il faut traiter l'événement associé.
- La méthode `setOnAction()` du bouton permet d'enregistrer un *Event Handler* (c'est une interface fonctionnelle possédant la méthode `handle(event)` qui définit l'action à effectuer).

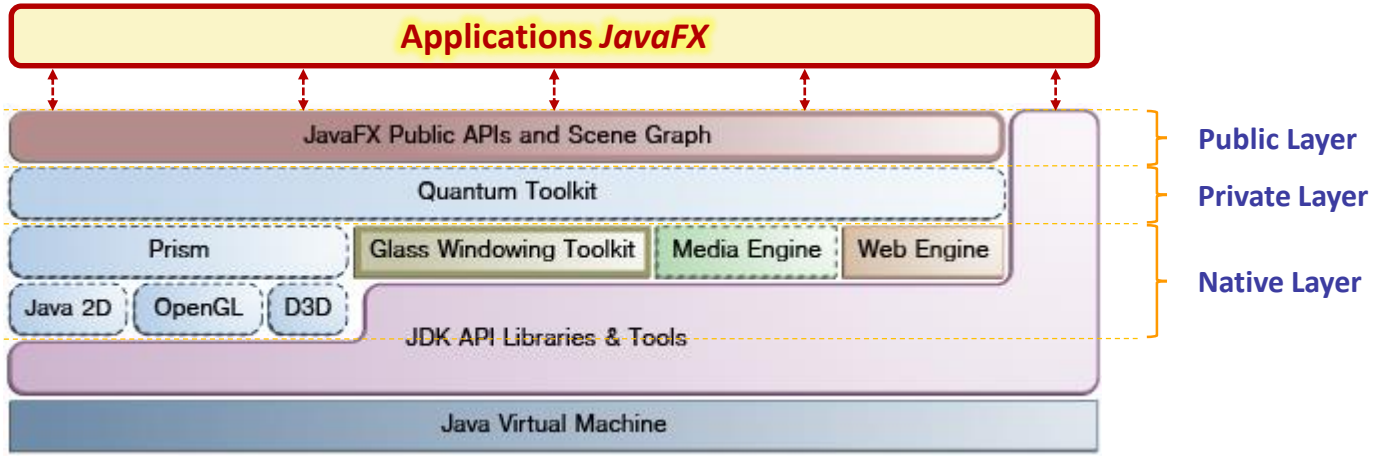
```
public void start(Stage premierStage) {
  premierStage.setTitle("My First JavaFX App");
  BorderPane root = new BorderPane();
  Button btnHello = new Button("Say Hello");

  btnHello.setOnAction(
                    event -> System.out.println("Hello World !"));

  root.setCenter(btnHello);
  Scene scene = new Scene(root, 250, 100);
  premierStage.setScene(scene);
  premierStage.show();
}
```



Architecture *JavaFX*
Concepts techniques

Architecture technique [1]

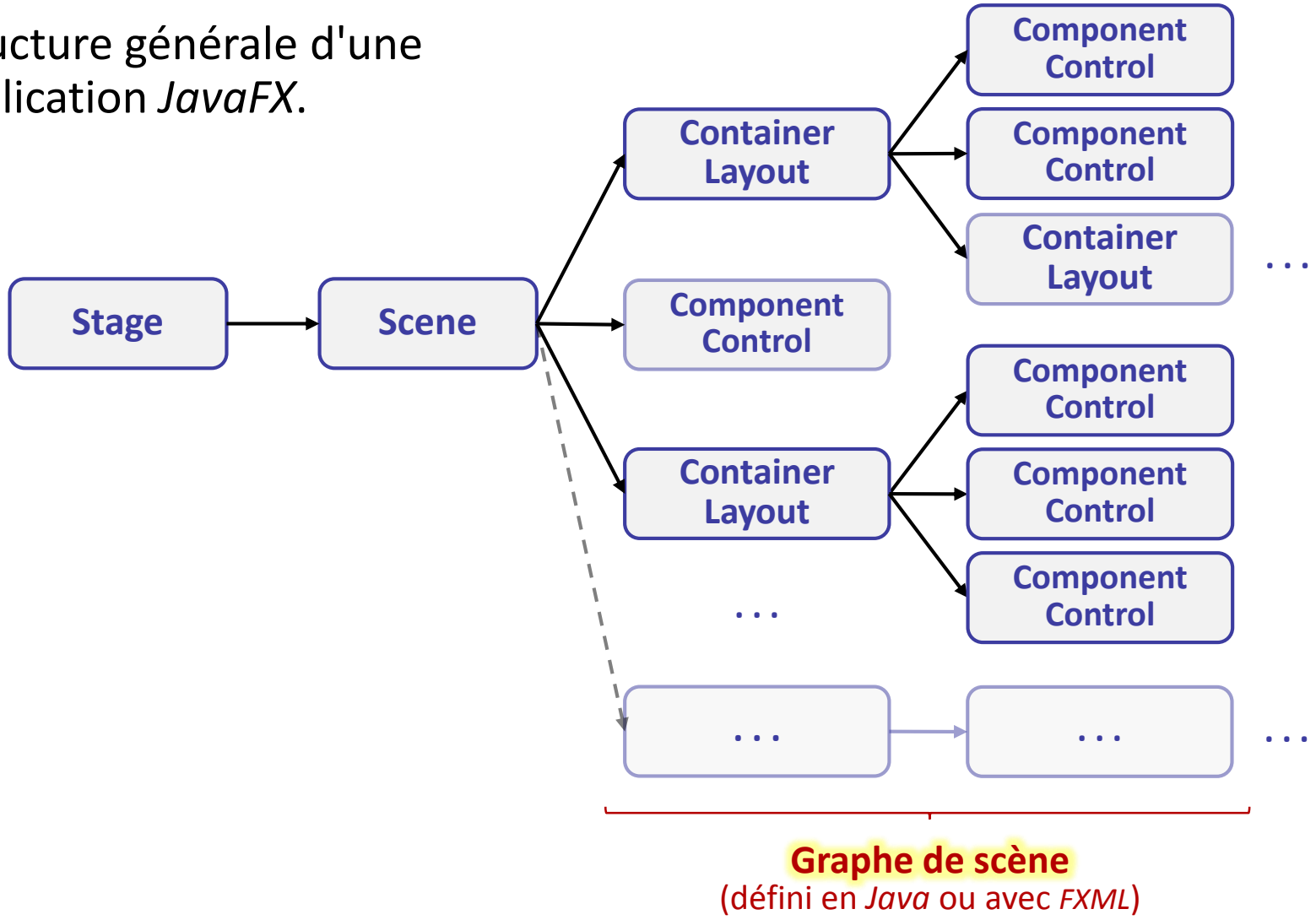
- L'architecture technique de la plateforme *JavaFX* est composée de plusieurs couches *(library stack)* qui reposent sur la machine virtuelle *Java (JVM)*.



- 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).
- Les "briques" intermédiaires ne seront donc pas décrites dans ce cours.

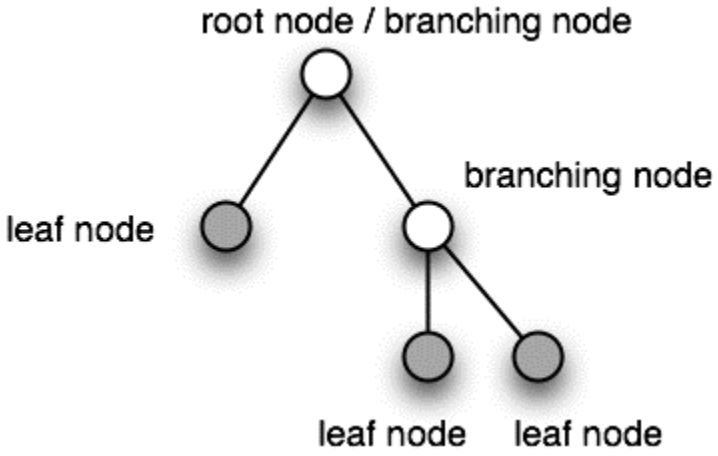
Éléments d'une application

- Structure générale d'une application *JavaFX*.



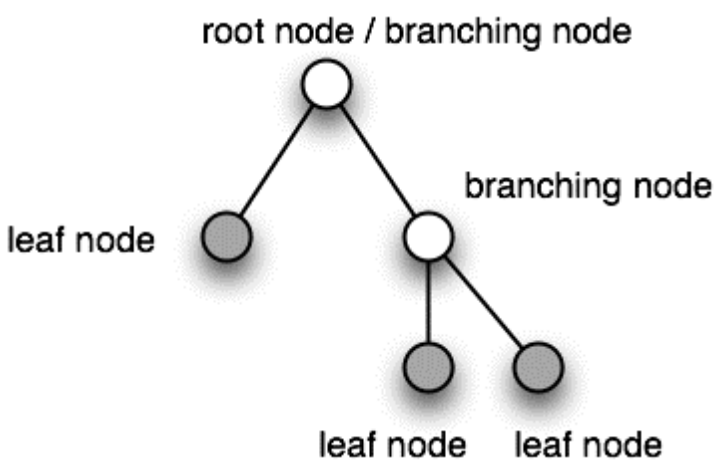
Graphe de scène [1]

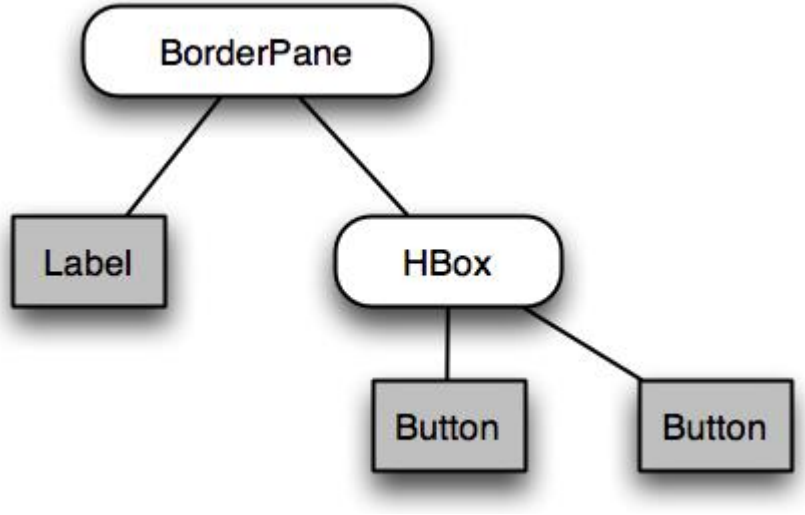
- Le **graphe de scène** ***(scene graph)*** est une notion importante qui représente la structure hiérarchique de l'interface graphique.
- Techniquement, c'est un graphe acyclique orienté (arbre orienté) avec :
  - une racine *(root)*
  - des nœuds *(nodes)*
  - des arcs qui représentent les relations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



Graphe de scène [2]

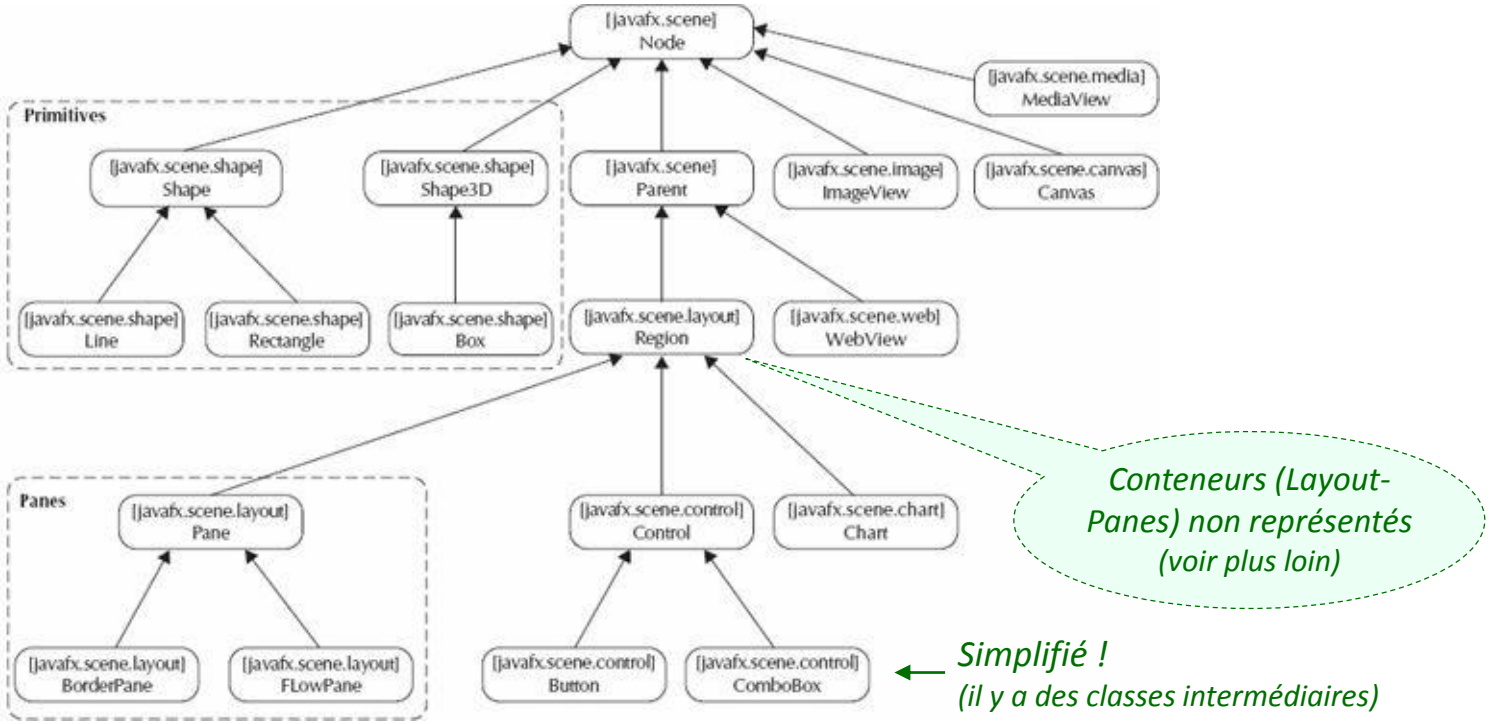
- Les **feuilles** de l'arbre sont généralement constitués de composants visibles (boutons, champs texte, …) et les **nœuds intermédiaires** (y compris la **racine**) sont généralement des éléments de structuration (souvent invisibles), typiquement des conteneurs ou panneaux de différents types (*HBox,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





Graphe de scène [3]

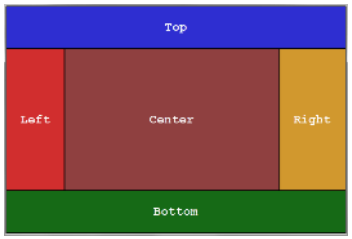
- Tous les éléments contenus dans un graphe de scène sont des objets qui ont pour classe parente la classe **Node**.
- La classe **Node** comporte de nombreuses sous-classes :



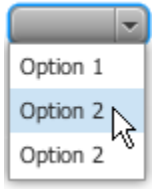
Graphe de scène [4]

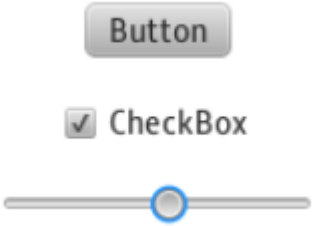
- Parmi les sous-classes de **Node** on distingue différentes familles :
  - Les **formes primitives** (***Shape***) 2D et 3D
    - ⇨ `Line`, `Circle`, `Rectangle`, `Box`, `Cylinder`, …

  - Les **conteneurs** (***Layout-Pane***) qui se chargent de la disposition *(layout)* des composants enfants et qui ont comme classe parente **Pane**.
    - ⇨ `AnchorPane`, `BorderPane`, `GridPane`, `HBox`, `VBox`, …



  - Les **composants standard** (***Controls***) qui étendent la classe **`Control`**.
    - ⇨ `Label`, `Button`, `TextField`, `ComboBox`, …

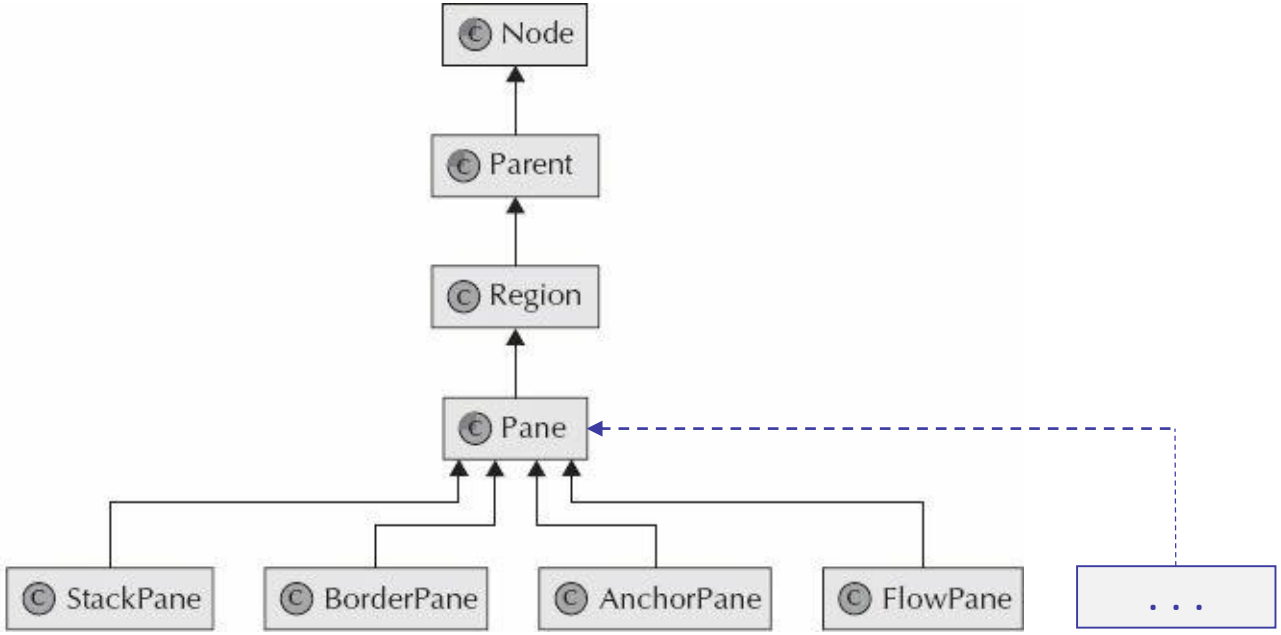




  - Les **composants spécialisés** qui sont dédiés à un domaine particulier (par exemple : lecteur multimédia, navigateur web, etc.).
    - ⇨ `MediaView`, `WebView`, `ImageView`, `Canvas`, `Chart`, …

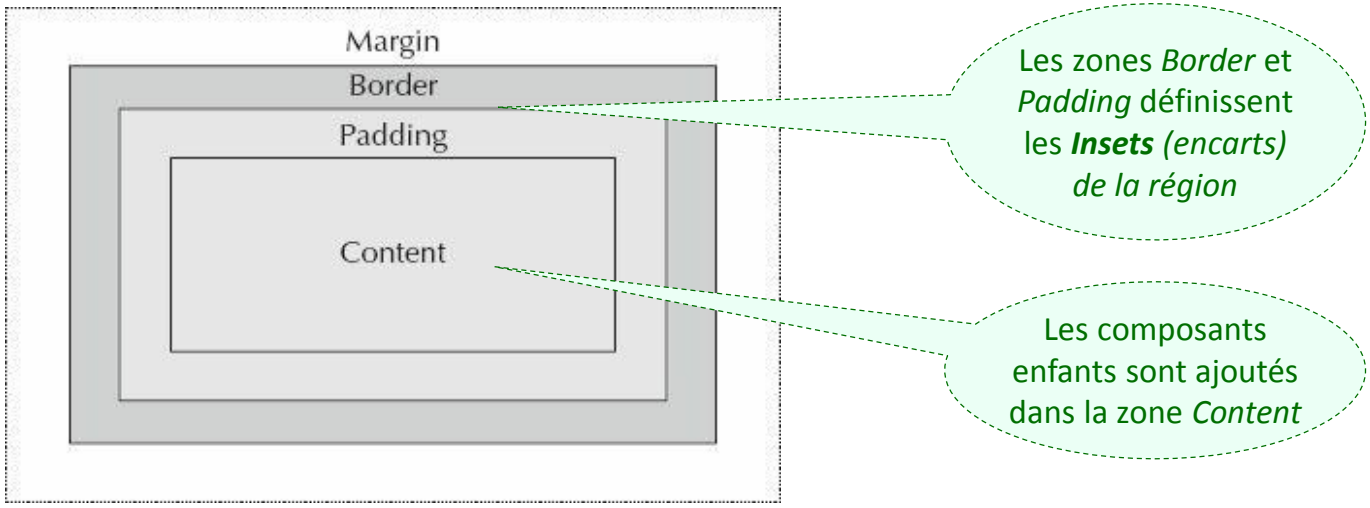
Conteneurs

- 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 [1]

- La classe **`Region`** est la classe parente des composants *(Controls)* et des conteneurs *(Layout-Panes)*.
- Elle définit des propriétés qui affectent la représentation visuelle.
- Les différentes zones d'une région sont basées sur la spécification du *Box-Model* CSS 3 (selon normalisation du W3C).
  - Elles définissent les notions *Margin, Border, Padding, Insets, Content*



Region - Structure visuelle [2]

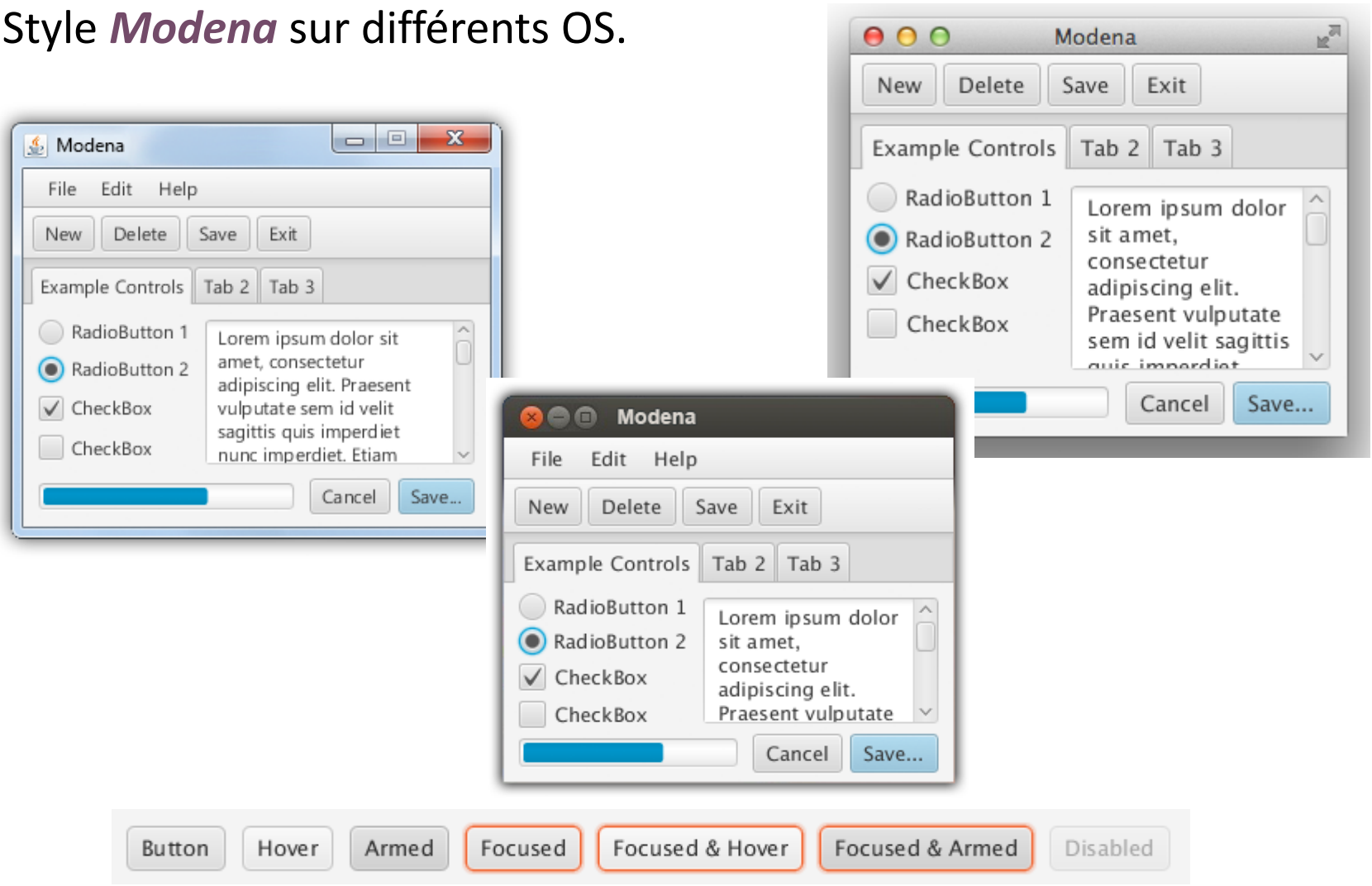
- Comme la classe `Region` est la classe parente de tous les conteneurs et de tous les composants, ses propriétés sont héritées et peuvent s'appliquer à une grande variété d'éléments qui sont utilisés pour créer les interfaces.
- Parmi ces propriétés, on peut mentionner :
  - `border`
    - ⇨ Bordure autour de la région
  - `background`
    - ⇨ Couleur ou image d'arrière-plan de la région
  - `padding`
    - ⇨ Marge (espace) autour du contenu de la région
- L'utilisation et le codage des propriétés `border` et `background` sont expliqués plus en détail dans le chapitre suivant consacré aux conteneurs et *layout-panes* (à la section *Disposition des composants - Généralités*).

Style – 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 défaut depuis *JavaFX 8*
  - STYLESHEET_CASPIAN : A été défini pour *JavaFX 2*

Style – Look and Feel [2]

- Style ***Modena*** sur différents OS.



Style – Look and Feel [3]

- 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 :

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

- Des **styles externes** *(third-party)* peuvent également être importés et appliqués. On trouve différentes réalisations, par exemple :
  - Apple *Aqua* (AquaFX)
  - Microsoft *Modern UI* (JMetro) *Windows-7 Aero* (AeroFX)
  - Twitter *Bootstrap* (Fextile)
  - Flatter *(Embedded UI)*, . . .