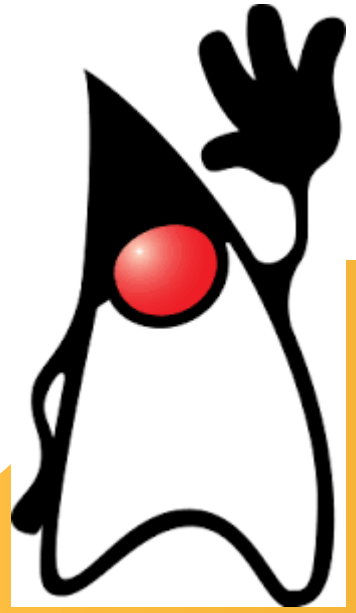
RÉPUBLIQUE TUNISIENNE
MESRS
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

SFAX



Cours Programmation Orientée Objets Avancée

Préparé par :
Hedi HMANI, Manel BEN SALAH, Sameh HBAIEB TURKI

Année universitaire
2020-2021

Chapitre 3: Interfaces graphiques avec JavaFX

JavaFX

Ce que vous allez apprendre:

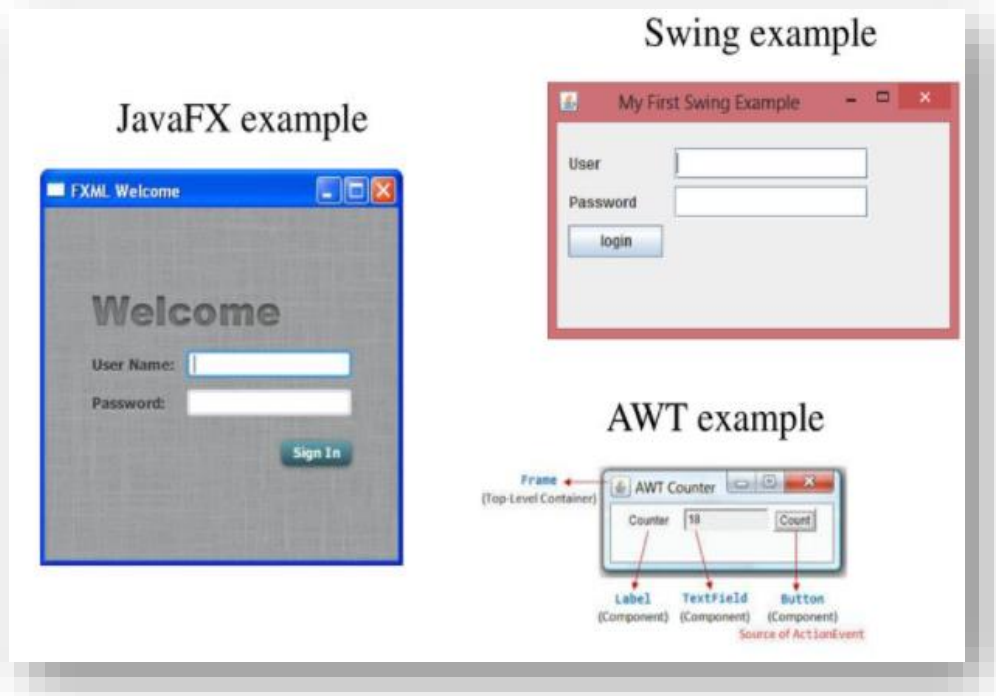
A la fin de ce chapitre, vous allez apprendre à:

- Concevoir et développer une application JavaFX
- Manipuler les différentes layouts et composants graphiques de base
- Personnaliser une application JavaFX avec CSS

Présentation de JavaFX

Bref historique
Les API Graphiques en java

- **Java AWT**: apparue avec java 1, natif. Les composants AWT sont trop dépendants de la plateforme java
- **Java Swing**: apparue avec java 2, basée sur AWT sans ressources
- **JavaFX**: devient le standard officiel pour le développement des interfaces des applications Java avec java 8, mixant java et XML (FXML)



Présentation de JavaFX

- JavaFX est un ensemble de bibliothèque java pour la création des applications graphiques clientes. Il est issue du projet OpenJFX
- JavaFX est utilisée pour concevoir, créer, tester et déployer des applications GUI multiplateformes
- JavaFX intègre une boîte à outils très riche permettant de concevoir des graphiques 2D + 3D, de l'audio, de la vidéo et des applications Web.
- JavaFx est bien adapté pour la création des applications desktop, des applications web, mobiles ou aussi embarquées.

Présentation de JavaFX

- Parmi les avantages de JavaFX est la possibilité de concevoir des applications riches de type RIA (Rich Interface Application) proposant des fonctionnalités étendues notamment :
  - Des composants graphiques évolués sont proposés (barre de menu, onglets, treeview, grille de données, ...)
  - Support du drag and drop
  - Un enrichissement des fonctionnalités graphiques notamment grâce à des effets visuels et une intégration forte du multimédia

Les solutions concurrentes

Adobe Flex/AIR

La solution d'Adobe permet aussi bien la création de RIA (Flex) que de RDA (Rich Desktop Application). Cette technologie est principalement basée sur Flash et XML. (framework payant)

Microsoft Silverlight

la solution de Microsoft permet, elle aussi, de développer des RIA et des RDA. La conception d'interfaces riches s'effectue aussi en XML et plus particulièrement en XAML. Tout comme Flex, la conception d'interfaces riches est facilité par la souplesse et la simplicité de XML. (framework payant)

AJAX

AJAX est la technologie permettant de concevoir des RIA (JavaScript + XML + CSS). L'utilisation d'AJAX passe le plus souvent par l'intermédiaire d'un framework (c’est gratuit), comme:Dojo, GWT, ZK, ….

Programmation procédurale VS déclarative

- Avec JavaFX, les interfaces peuvent être créées de deux manières :
  - **Procédurale** : en écrivant du code Java qui fait appel aux API de la plateforme et qui utilise les composants/conteneurs à disposition (classes et interfaces)
  - **Déclarative** : en décrivant l'interface dans un fichier au format FXML qui sera ensuite chargé dynamiquement dans l'application
- La première partie du cours vise à décrire les bases de la technique procédurale (programmatique) permettant de créer des interfaces.
- Et la deuxième partie abordera la deuxième possibilité de créer les interfaces (les vues) en utilisant notamment l'outil SceneBuilder qui permet, de manière interactive, de créer les fichiers FXML.

Les modules

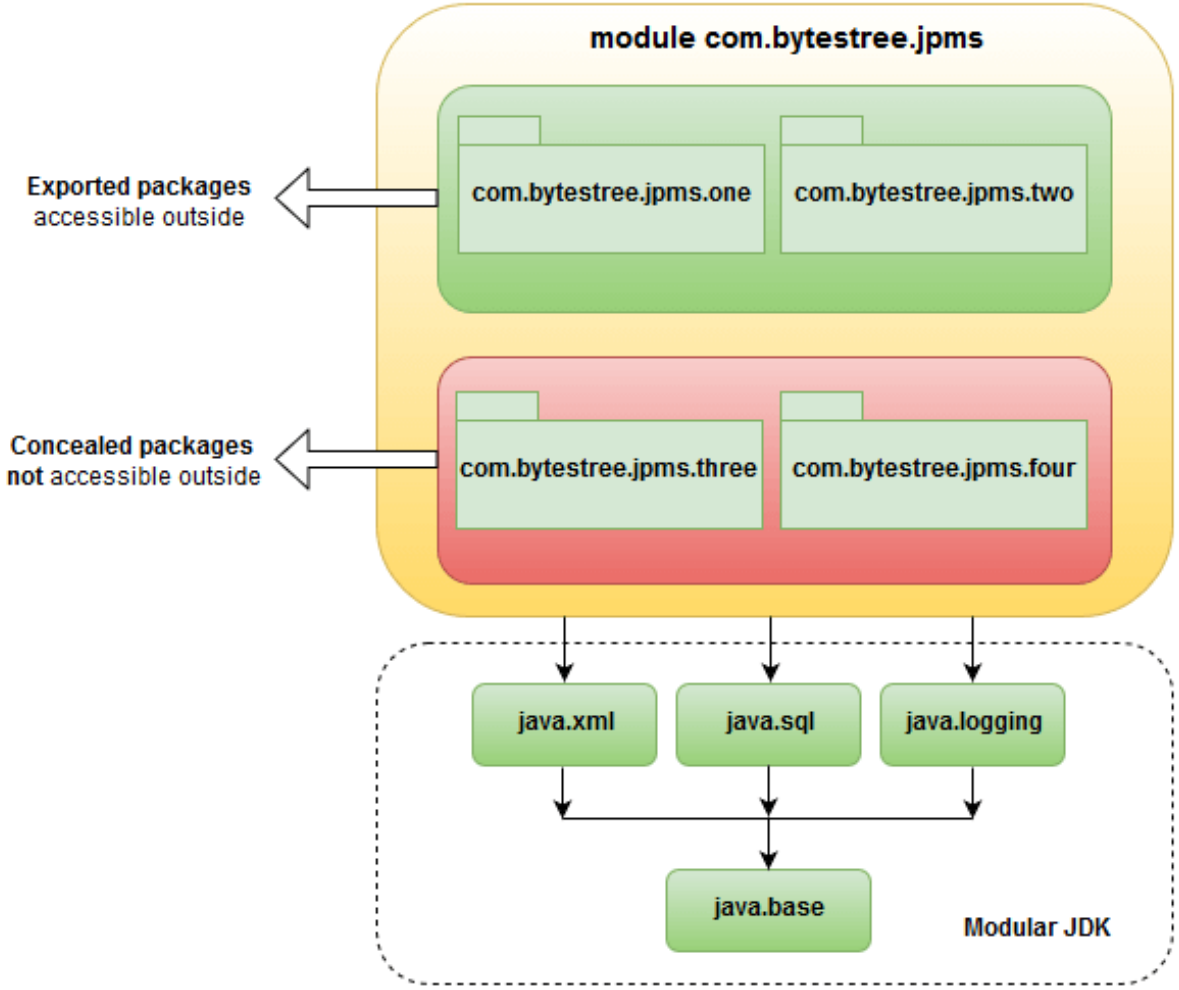
Notion de module en Java 9

- La notion de module est introduite en Java 9
- Un module est (généralement) un fichier JAR qui contient un ensemble de packages. Tout comme les champs et les méthodes sont regroupés en classes, et à leur tour en packages, les packages sont regroupés en modules.
- Modulariser une application signifie la décomposer en plusieurs modules qui fonctionnent ensemble: définir les dépendances (export/require) entre les modules.
- Avec les modules, il est nécessaire de définir, pour chaque application (déployée en fichier JAR), l'ensemble des modules dont elle dépend (**require**), ainsi que les packages de l'application qui peuvent être utilisés par d'autres modules (**export**).

Notion de module en Java 9

- Chaque module doit définir à sa racine un fichier qui s'appelle impérativement **module-info.java**. Ce fichier contient une définition déclarative du module:
  - Name – le nom du module
  - Dependencies – liste des modules dont dépend le module courant (requires)
  - Public Packages – liste des packages qui sont accessibles en dehors du module courant (export)
  - Services Offered – Les services qui peuvent être consommés par d'autres modules
  - Services Consumed – Les services consommés par le module courant

Notion de module en Java 9



Java 9 Module

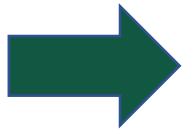
```
module com.bytestree.jpms {
    exports com.bytestree.jpms.one;
    exports com.bytestree.jpms.two;
    requires java.sql;
    requires java.logging;
    requires java.xml;
}
```

Notion de module en Java 9

- Exemple : Création du module hello.modules

Création de package dans le module hello.modules

```
package com.baeldung.modules.hello;

public class HelloModules {
    public static void doSomething() {
        System.out.println("Hello, Modules!");
    }
}
```



Exportation du package dans le fichier module.info

```
module hello.modules {
    exports com.baeldung.modules.hello;
}
```

Notion de module en Java 9

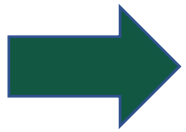
- Exemple (Suite) : Création du module main.app qui utilise de hello.modules

Création du module main.app

```
package com.baeldung.modules.main;

import com.baeldung.modules.hello.HelloModules;

public class MainApp {
    public static void main(String[] args) {
        HelloModules.doSomething();
    }
}
```

Classe définie dans le package

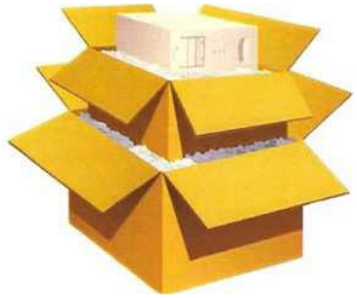
Package du module hello.modules



Fichier module.info

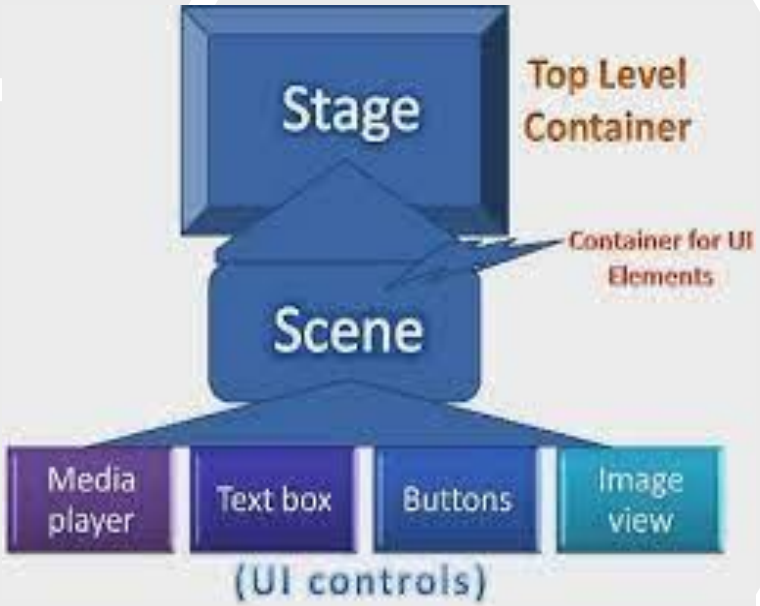
```
module main.app {
    requires hello.modules;
}
```

Le modèle hiérarchique de l’interface



Architecture d’une interface JavaFX

- 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** 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**



Première Application JavaFX

```
public class Main 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); }
}
```

Héritage de la classe Application

Création du cadre principal de la fenêtre

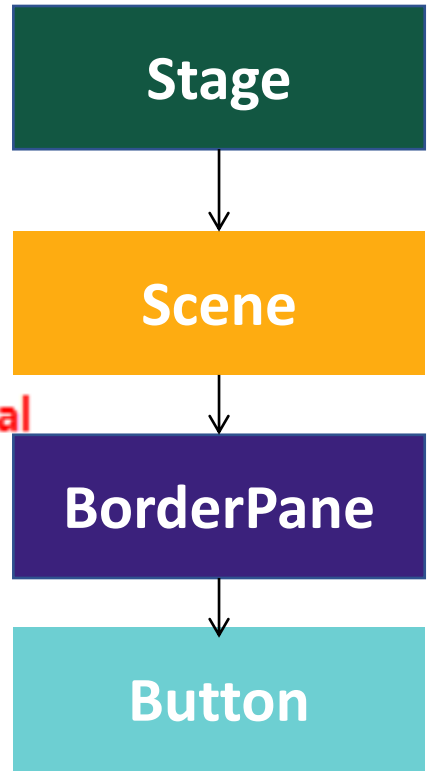
Utiliser le layout BorderPane

Créer le conteneur principal scene

Associer le conteneur principal scene au cadre principal

Afficher la fenêtre

Exécuter l'application et lancer la fenêtre



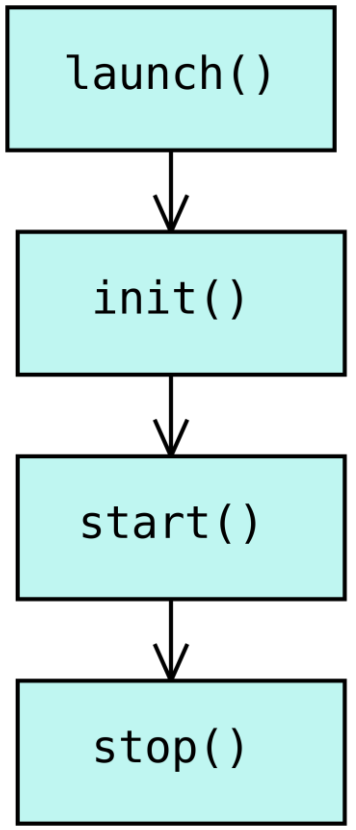
Cycle de vie d'une application JavaFX

- Pour afficher la fenêtre principale (Stage), il faut 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();
}
```

Cycle de vie d'une application JavaFX

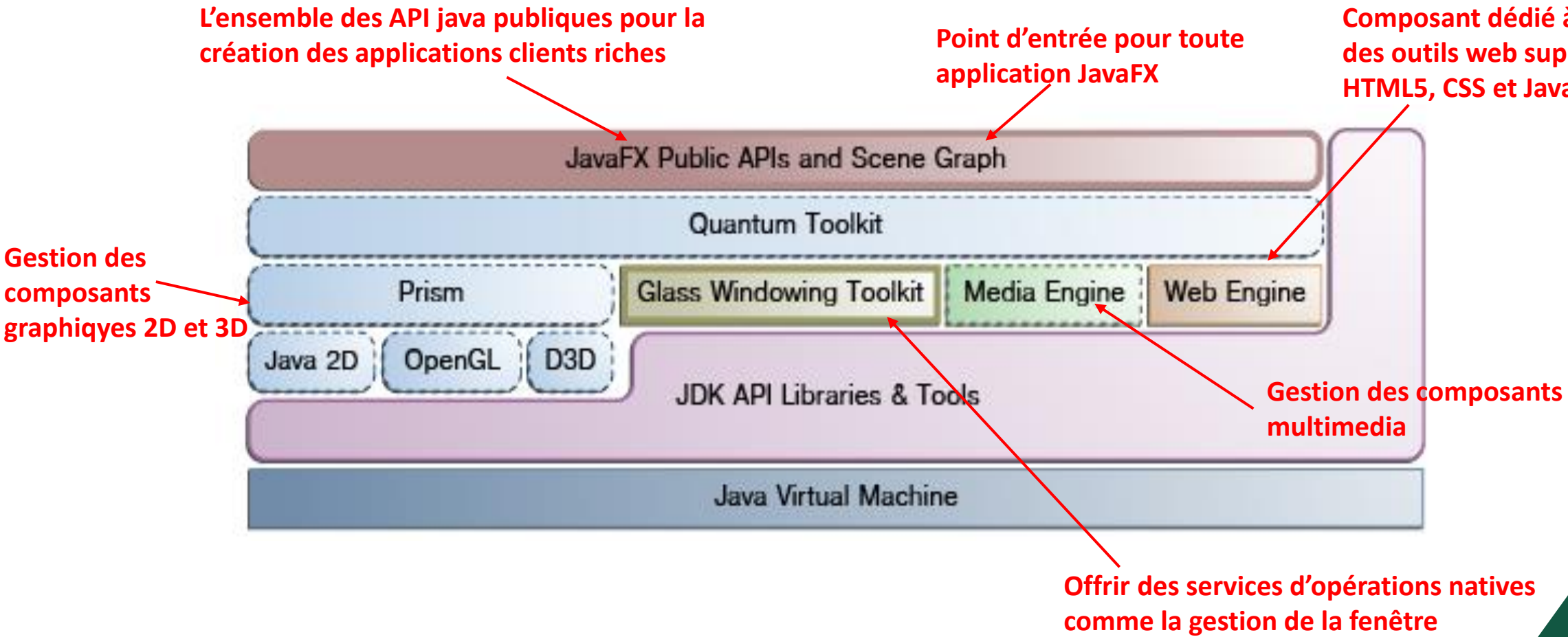
- Le point d'entrée d'une application JavaFX est constitué de l'instance de la classe **Application** (généralement une sous-classe).
- Lors du lancement d'une application par la méthode statique **Application.launch()** le runtime JavaFX effectue les opérations suivantes :



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]). La méthode start() est une méthode abstraite dans la classe Application qu'il faut la redéfinir.
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()** lorsque l'application se termine

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 API publiques du framework JavaFX

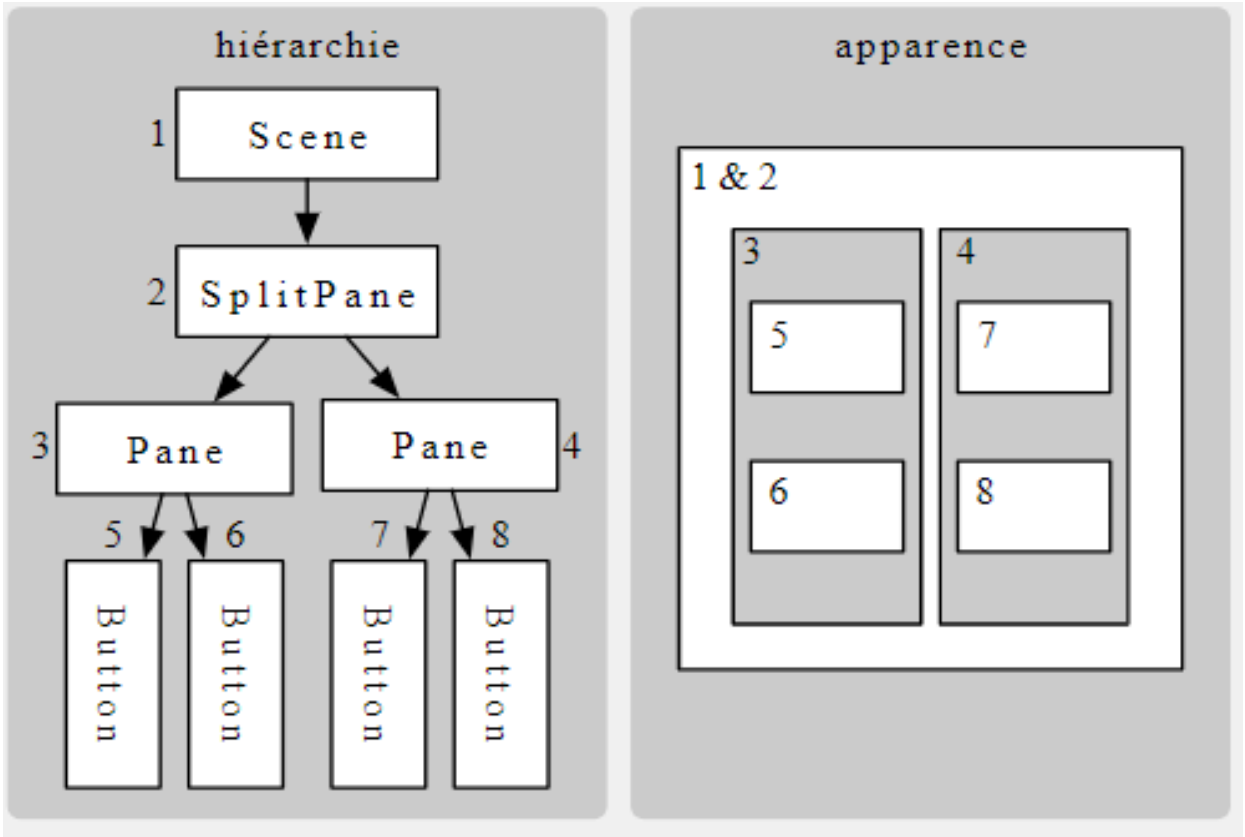
javafx.base	Définit l'API de base
javafx.controls	Définit la majorité des composants graphiques de l'API
javafx.fxml	Définit l'API relative au langage FXML qui permet de décrire une interface utilisateur d'une manière alternative à l'écriture de lignes de code
javafx.graphics	Définit l'API relative aux conteneurs, animations, effets visuels, formes 2D et 3D, images, impression, fenêtres, événements, robots, au support du CSS et à l'application
javafx.media	Définit l'API dédié à la lecture de contenu audio et vidéo
javafx.swing	Définit l'API qui fournit le support d'interopérabilité entre JavaFX et Swing
javafx.web	Définit l'API dédié à l'affichage de contenu web (notamment un éditeur HTML et un moteur de rendu de pages web basé sur WebKit)

Le graphe de scène

- Une interface graphique JavaFX est constituée d'un certain nombre de nœuds (nodes), qui peuvent être de différente nature : simples formes géométriques (cercles, lignes, etc.), composants d'interface utilisateur (bouton, menu, etc.), conteneurs, etc.
- Ces nœuds sont organisés en une hiérarchie que l'on nomme le **graphe de scène** (scene graph).
- Le graphe de scène est un simple arbre qui reflète l'organisation des nœuds à l'écran
- Chaque nœud a au plus un parent.
- Les noms des nœuds du graphe de scène sont ceux des classes JavaFX correspondant à ce type de nœud

Le graphe de scène

- Exemple:



Graphe de scène

Fenêtre JavaFX

Le graphe de scène

Comme cet exemple l'illustre, on peut distinguer plusieurs genres de nœuds JavaFX :

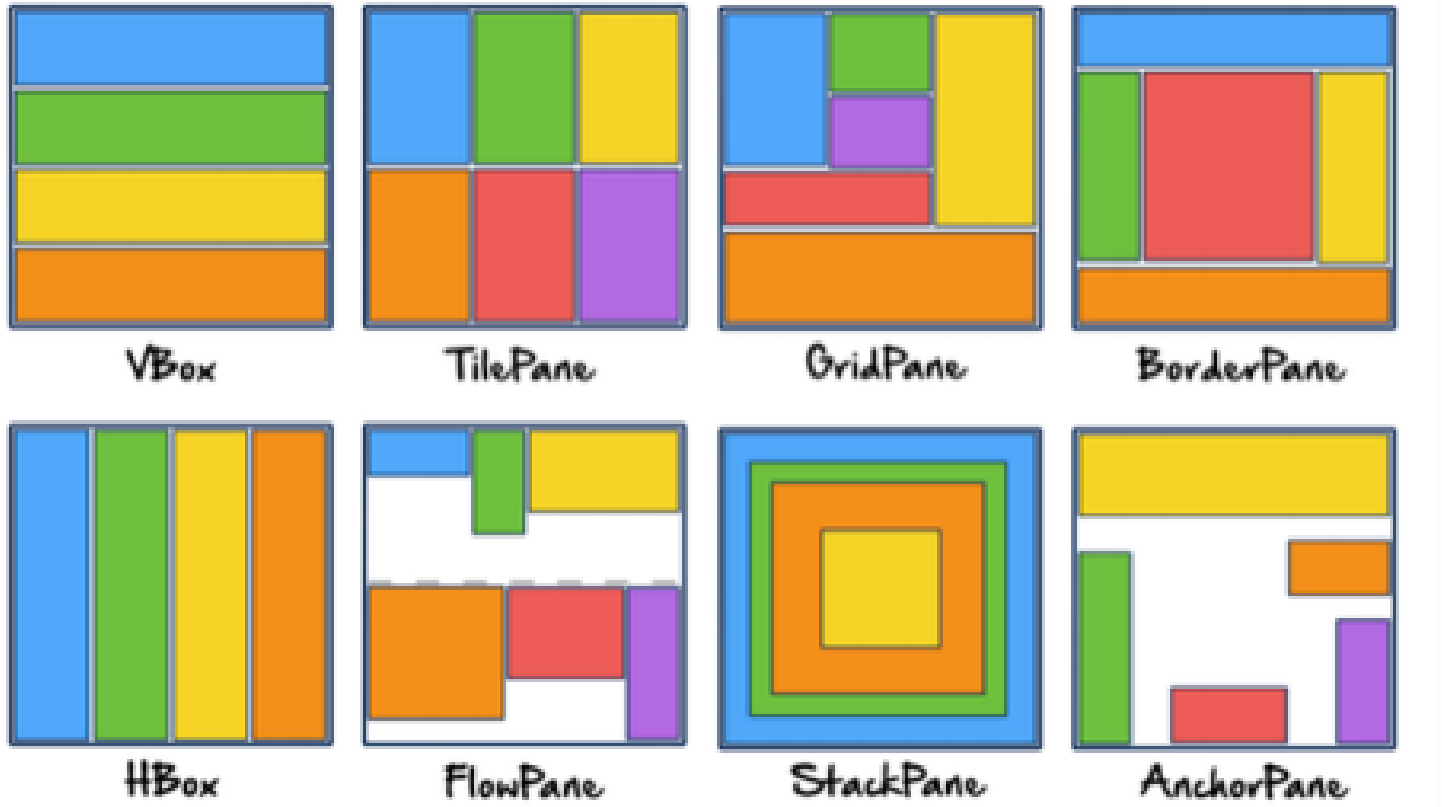
- **les nœuds de base:** qui représentent les feuilles de l'arbre et sont généralement constitués de composants visibles (boutons, champs texte, …)
- **les nœuds intermédiaires:** qui possèdent un certain nombre de nœuds enfants et qui sont généralement des éléments de structuration (souvent invisibles), typiquement des conteneurs de différents types (HBox, VBox, BorderPane, …)
- **la racine**, qui doit toujours avoir le type Scene

Les conteneurs de disposition (Layout)

Qu’est ce qu’un Layout ?

- Un Layout (container ou gestionnaire de mise en page) est un conteneur graphique qui hérite de la classe javafx.scene.layout.Pane.
- Il permet d’afficher un groupe de Contrôles UI (du package javafx.scene.control) (ou d’autres containers) suivant une disposition qui lui est propre.

Les types de conteneurs



Les types de conteneurs

BorderPane permet de diviser une zone graphique en cinq parties : top, down, right, left et center.

Hbox permet d'organiser une série de nœuds en une seule rangée.

Vbox permet d'aligner verticalement les éléments graphiques.

StackPane tous les nœuds sont dans une pile unique avec chaque nouveau nœud ajouté au dessus du nœud précédent

Les types de conteneurs

GridPane permet de créer une grille d'éléments organisés en lignes et en colonnes

FlowPane les composants sont placés les uns derrière les autres. Commencer une nouvelle rangée de composants si on ne peut pas les faire tenir dans une seule rangée.

TilePane similaire au FlowPane, chacune de ses cellules fait la même taille.

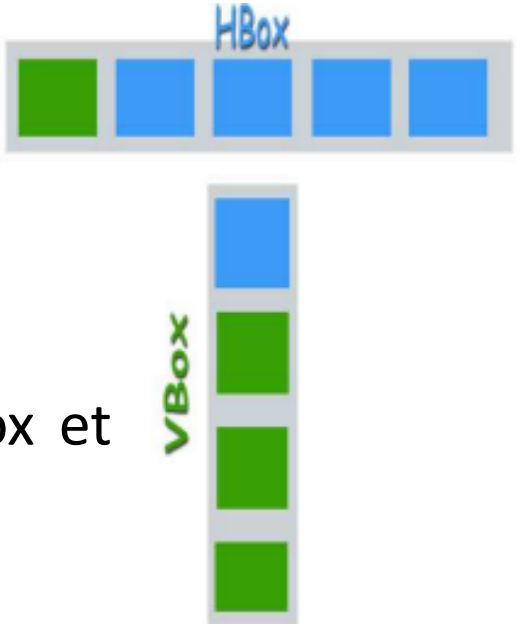
AnchorPane permet de fixer un élément graphique par rapport à un des bords de la fenêtre : top, bottom, right et left

BorderPane

- Le conteneur BorderPane du package **javafx.scene.layout** permet de placer les composants enfants dans cinq zones : Top, Bottom, Left, Right et Center.
- Un seul objet Node(composant, conteneur, …) peut être placé dans chacun de ces emplacements.
- BorderPane, propose les méthodes suivantes :
  - **setCenter**(Nodes),
  - **setLeft**(Nodes),
  - **setRight**(Nodes),
  - **setTop**(Nodes),
  - **setBottom**(Nodes)

HBox/VBox

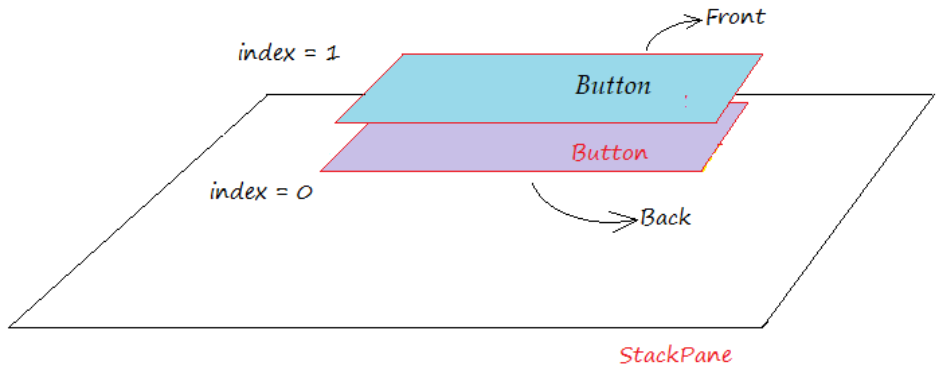
- Le layout Hbox place les composants sur une ligne horizontale : Les composants sont ajoutés à la suite les uns des autres (de gauche à droite).
- Le layout Vbox place les composants sur une colonne verticale : composants sont ajoutés à la suite les uns des autres (de haut en bas).
- Hbox et VBox mettent à disposition les méthodes suivantes :
  - **setPadding**(inset) permet de régler l'espace entre le bord du hbox et des nœuds.
  - **setSpacing**(int) régle l'espace entre les nœuds.
  - **getChildren().add**(nodes) permet d'ajouter un nœuds au Hbox

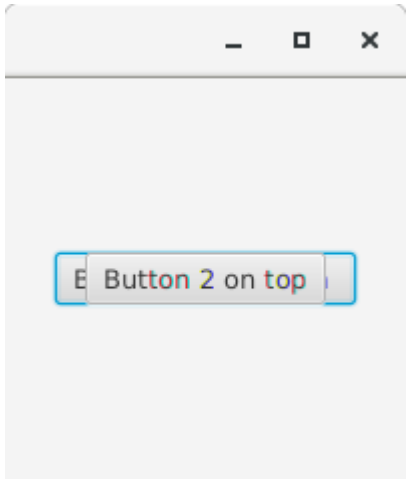


StackPane

- Le conteneur **StackPane** empile les composants enfants les uns au dessus des autres dans l'ordre d'insertion : les premiers "au fond", les derniers "au-dessus" (back-to-front).

```
@Override
   public void start(Stage primaryStage) throws Exception
{
        Button btn1 = new Button("Button 1 on bottom ");
        Button btn2 = new Button("Button 2 on top");
        StackPane root = new StackPane();
        Scene scene = new Scene(root,200,200);
        // Add Button to StackPane
        root.getChildren().addAll(btn1,btn2);
        // ou bien
        // stackPane.getChildren().add(btn1);
        // stackPane.getChildren().add(btn2);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```



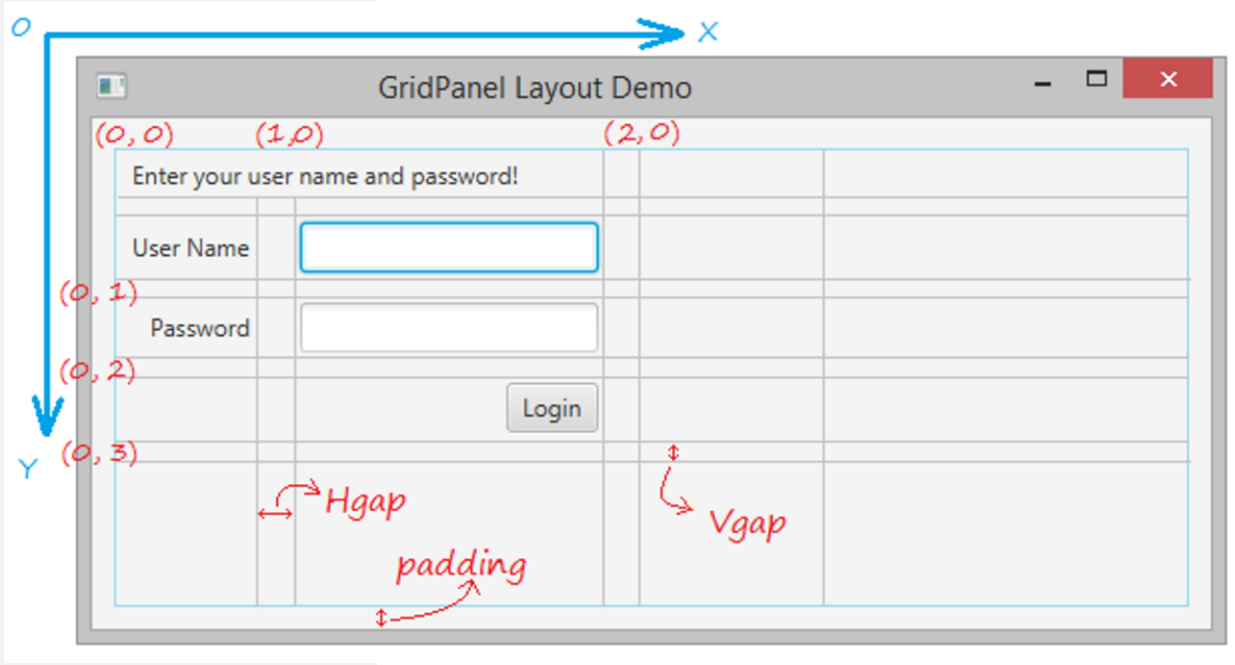


GridPane

- Le conteneur GridPane permet de disposer les composants enfants dans une grille flexible (arrangement en lignes et en colonnes), un peu à la manière d'une table HTML.
- La grille peut être irrégulière, la hauteur des lignes et la largeur des colonnes de la grille ne sont pas nécessairement uniformes.
- La zone occupée par un composant peut s'étendre (span) sur plusieurs lignes et/ou sur plusieurs colonnes.
- Le nombre de lignes et de colonnes de la grille est déterminé automatiquement par les endroits où sont placés les composants.

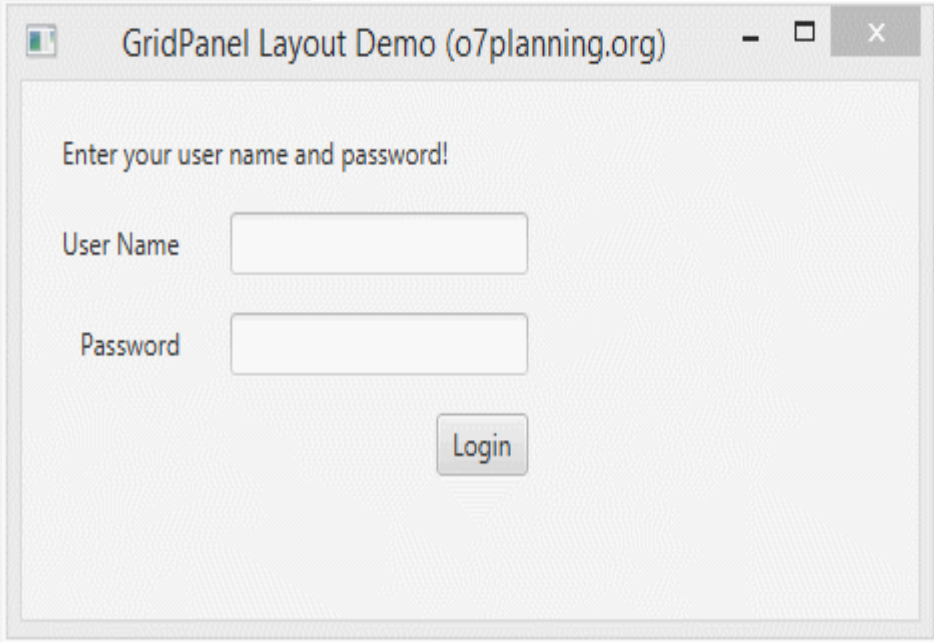
Exemple GridPane (1)

```
@Override
  public void start(Stage primaryStage) throws Exception {
    GridPane root = new GridPane();

    root.setPadding(new Insets(20));
    root.setHgap(25);
    root.setVgap(15);

    Label labelTitle = new Label("Enter your user name
            and password!");
    // Put on cell (0,0), span 2 column, 1 row.
    root.add(labelTitle, 0, 0, 2, 1);

    Label labelUserName = new Label("User Name");
    TextField fieldUserName = new TextField();
    Label labelPassword = new Label("Password");
    PasswordField fieldPassword = new PasswordField();

    Button loginButton = new Button("Login");
```



Exemple GridPane (2)

```
        GridPane.setHalignment(labelUserName, HPos.RIGHT);
        // Put on cell (0,1)
        root.add(labelUserName, 0, 1);
        GridPane.setHalignment(labelPassword, HPos.RIGHT);
        root.add(labelPassword, 0, 2);
        // Horizontal alignment for User Name field.
        GridPane.setHalignment(fieldUserName, HPos.LEFT);
        root.add(fieldUserName, 1, 1);
        // Horizontal alignment for Password field.
        GridPane.setHalignment(fieldPassword, HPos.LEFT);
        root.add(fieldPassword, 1, 2);
        // Horizontal alignment for Login button.
        GridPane.setHalignment(loginButton, HPos.RIGHT);
        root.add(loginButton, 1, 3);
        Scene scene = new Scene(root, 300, 300);
        primaryStage.setTitle("GridPanel Layout Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```



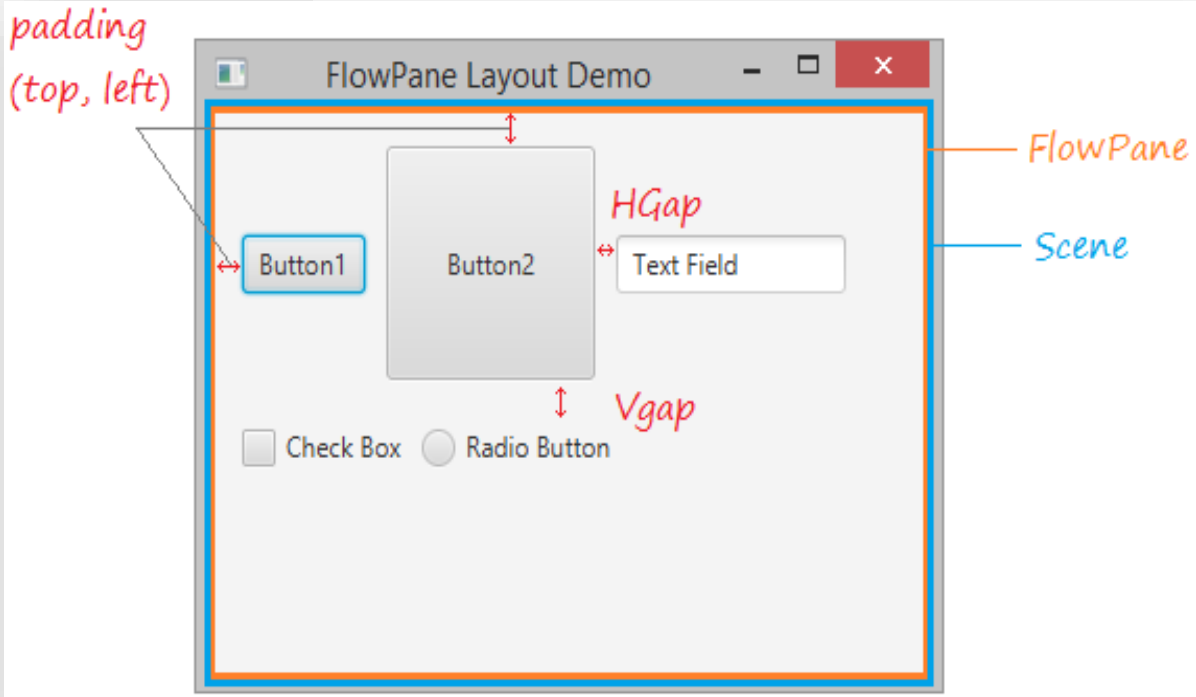
FlowPane

- Le layout FlowPane place les composants sur une ligne horizontale ou verticale et passe à la ligne ou à la colonne suivante (wrapping) lorsqu'il n'y a plus assez de place disponible.
- Un des paramètres du constructeur (de type Orientation) détermine s'il s'agit d’un FlowPane horizontal (par défaut) ou vertical

Exemple FlowPane (1)

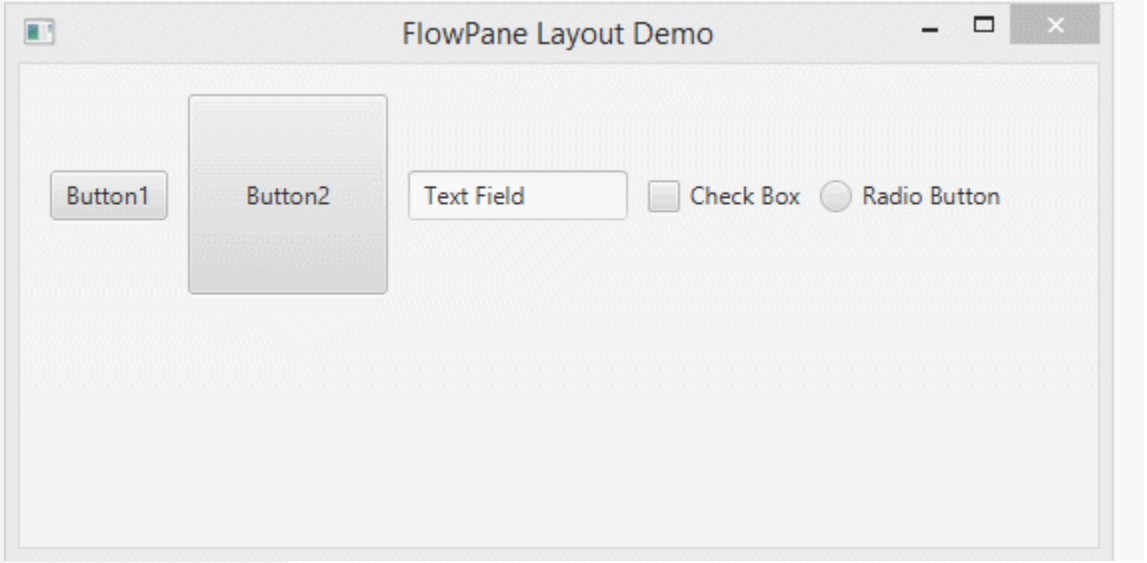
```
@Override
  public void start(Stage primaryStage) throws Exception {
    FlowPane root = new FlowPane();
     root.setHgap(10);
     root.setVgap(20);
     root.setPadding(new Insets(15,15,15,15));

     // Button 1
     Button button1= new Button("Button1");
     root.getChildren().add(button1);

     // Button 2
     Button button2 = new Button("Button2");
     button2.setPrefSize(100, 100);
     root.getChildren().add(button2);

     // TextField
     TextField textField = new TextField("Text Field");
     textField.setPrefWidth(110);
     root.getChildren().add(textField);
```

Insets(double top, double right, double bottom, double left)



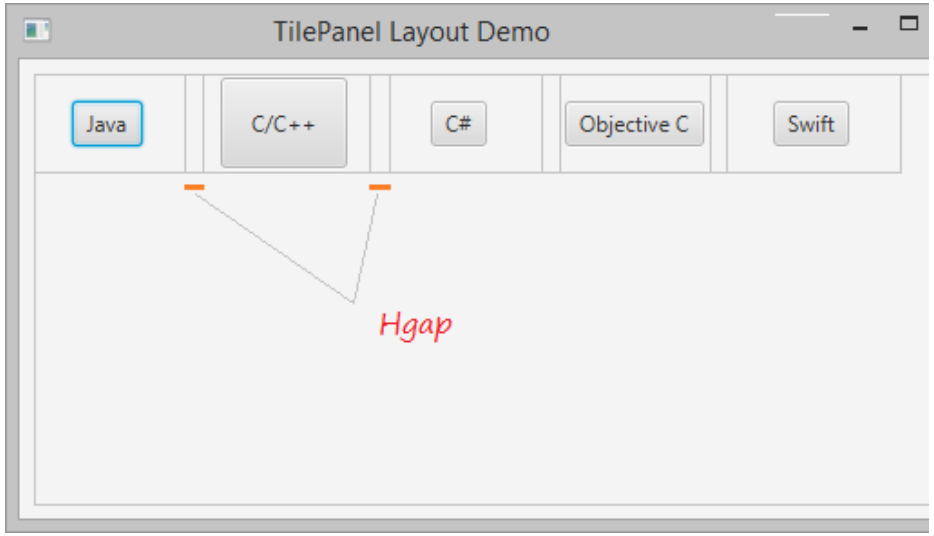
Exemple FlowPane (2)

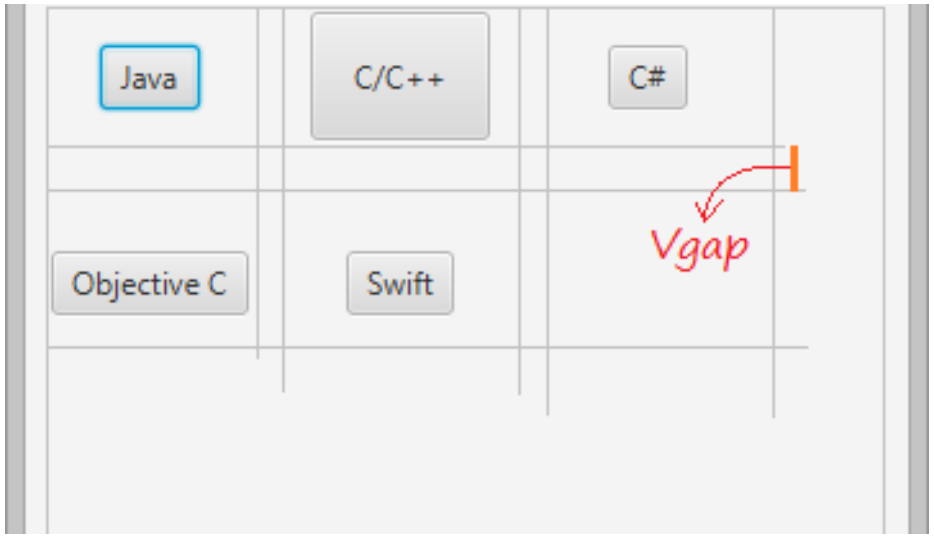
```
        // CheckBox
        CheckBox checkBox = new CheckBox("Check Box");
        root.getChildren().add(checkBox);

        // RadioButton
        RadioButton radioButton = new RadioButton("Radio Button");
        root.getChildren().add(radioButton);

        Scene scene = new Scene(root, 550, 250);

        primaryStage.setTitle("FlowPane Layout Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
}
```



TilePane

- Le layout TilePane place les composants dans une grille alimentée soit horizontalement (par ligne, de gauche à droite) soit verticalement (par colonne, de haut en bas).
- Un des paramètres du constructeur (de type Orientation) détermine s'il s'agit d'un TilePane horizontal (par défaut) ou vertical.
- On définit pour la grille un certain nombre de colonnes (propriété prefColumns) si l'orientation est horizontale ou un certain nombre de lignes (propriété prefRows) si l'orientation est verticale.
- Le conteneur TilePane est très proche du conteneur FlowPane. La différence principale réside dans le fait que toutes les cellules ont obligatoirement la même taille (ce qui n'est pas le cas pour FlowPane).





Les contrôles UI (les composants de base)

Les composants de base JavaFX

- Parmi les différents Node que l'on peut placer dans le graphe de Scène, les widgets sont les plus sophistiqués, ils intègrent des comportements évolués.
- De part sa nature flexible et ses performances graphiques, JavaFX possède beaucoup d'extensions (des bibliothèques supplémentaires de widgets).

Les composants de base JavaFX

Les composants de base JavaFX

- La création de widgets se fait toujours de la même façon :

1) On crée le widget, exemple :

```
Button button1 = new Button();
```

2) On paramètre le widget :

```
button1.setText("Créer un nouveau bouton");
```

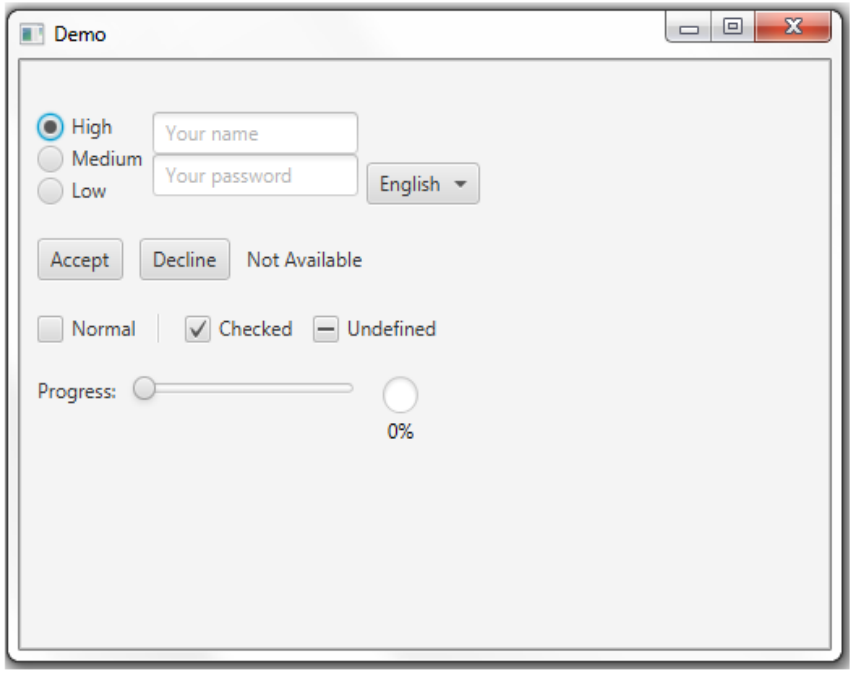
3) On l'ajoute à un container du scenegraph :

```
myHBox.getChildren().add(button1);
```

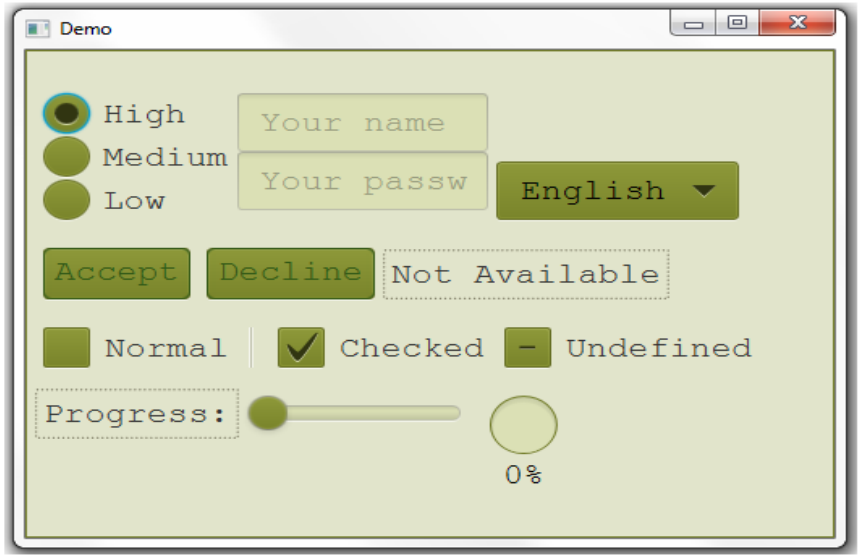
Les style sheets CSS

Utilisation de CSS avec JavaFX

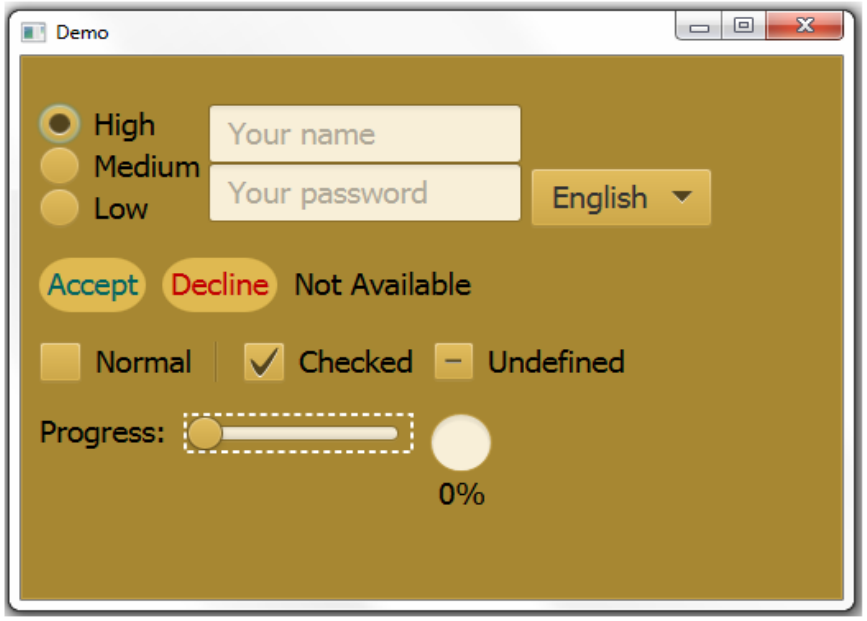
JavaFX with default style



controlStyle 1



controlStyle 2



Utilisation de CSS avec JavaFX

- L’utilisation de CSS dans les applications JavaFX est similaire à l’utilisation de CSS dans HTML.
- Les feuilles de style créées ont l’extension de **. css** et sont situées dans le même répertoire que la classe principale de l’application JavaFX

Utilisation de CSS avec JavaFX

controlStyle1.css

```
.root{
   -fx-font-size: 14pt;
   -fx-font-family: "Tahoma";
   -fx-base: #DFB951;
   -fx-background: #A78732;
   -fx-focus-color: #B6A678; }
.button1{
   -fx-text-fill: #006464;
   -fx-background-color: #DFB951;
   -fx-border-radius: 20;
   -fx-background-radius: 20;
   -fx-padding: 5; }
.button2{
   -fx-text-fill: #c10000;
   -fx-background-color: #DFB951;
   -fx-border-radius: 20;
   -fx-background-radius: 20;
   -fx-padding: 5; }
```

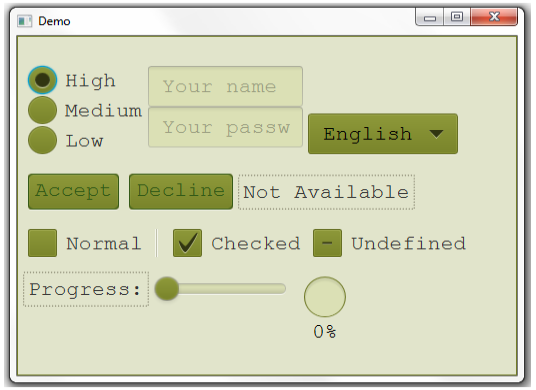
Class style

```
.slider{
   -fx-border-color: white;
   -fx-border-style: dashed;
   -fx-border-width: 2;
}
```

Id style

```
#font-button {
   -fx-font: bold italic 20pt "Arial";
   -fx-effect: dropshadow( one-pass-box , black , 8 , 0.0 , 2 , 0 );
}
```

JavaFX interface



Utilisation de CSS avec JavaFX

- Ajouter StyleScheet à l'application JavaFX:

```
Scene scene = new Scene(new Group(), 500, 400);
scene.getStylesheets().add("path/controlStyle1.css");
```

- Assigner un class style à un contrôle UI:

```
Button buttonAccept = new Button("Accept");
buttonAccept.getStyleClass().add("button1");
```

- Assigner un id style à un contrôle UI:

```
Button buttonFont = new Button("Font");
buttonFont.setId("font-button");
```

- Définir un style interne:

```
Button buttonColor = new Button("Color");
buttonColor.setStyle("-fx-background-color: slateblue; -fx-text-fill: white;");
```