

Développement d'application avec IHM

Les conteneurs



Christine Jacquin

La disposition des composants

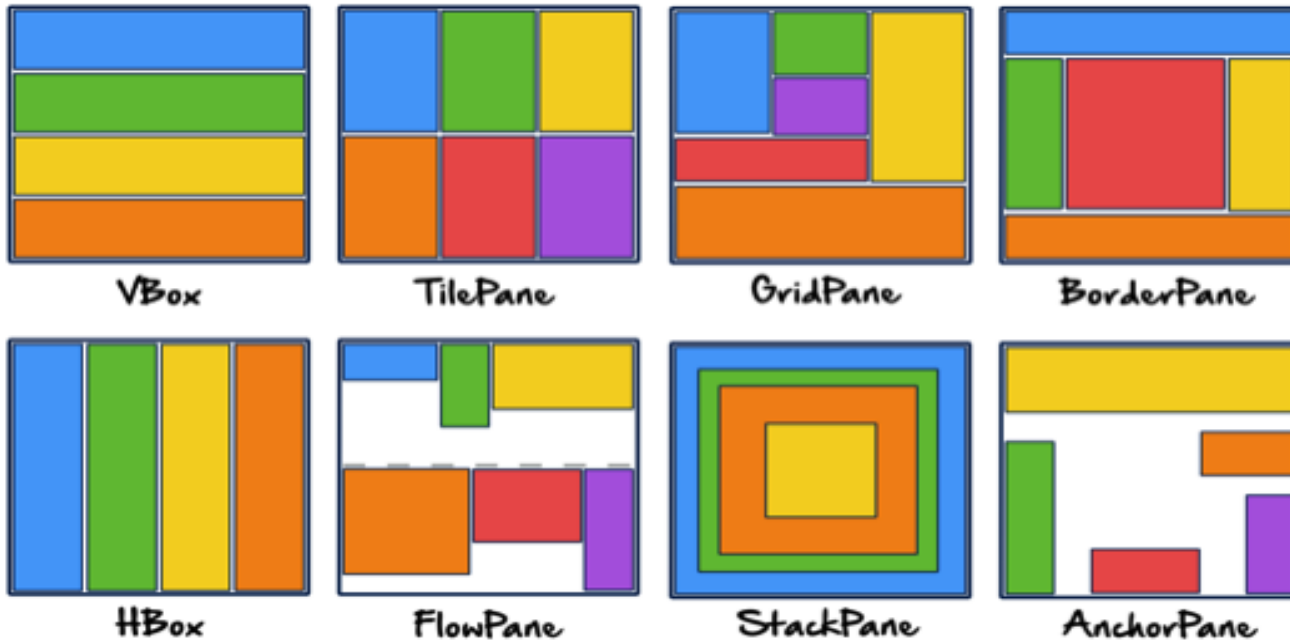
Il est possible en **JavaFX** de dimensionner et de positionner les composants de manière absolue (en pixels par exemple)

Mais ceci présente de nombreux inconvénients :

- La taille des composants peut varier, en fonction de :
 - la langue choisie (libellés, boutons, menus, ...)
 - la configuration de la machine cible
- La taille de la fenêtre affichée peut également varier :
 - en fonction des interactions de l'utilisateur (redimensionnement)
 - pour s'adapter à la résolution de l'écran de la machine cible (pour afficher l'intégralité de l'interface)

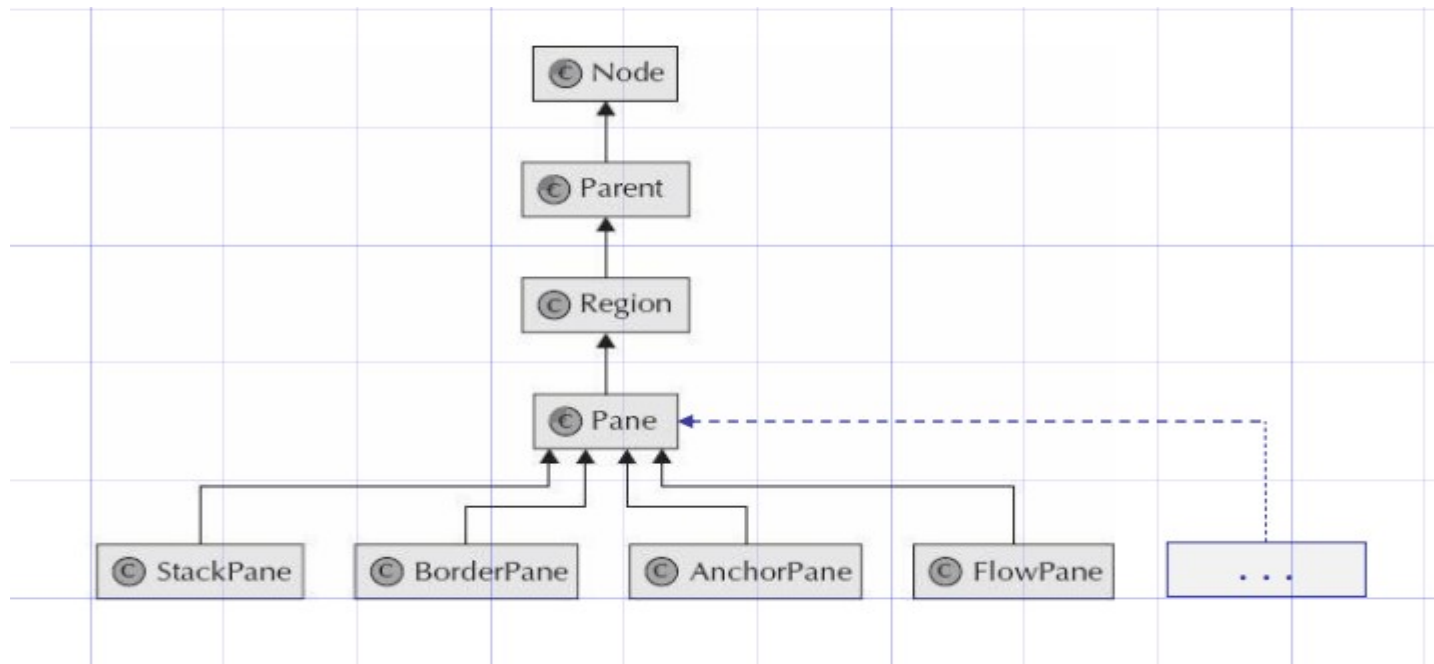
Les différents conteneurs

Dans un souci d'adaptation , nous utiliserons des conteneurs prédéfinis de JavaFX pour placer les composants dans l'interface.



Les conteneurs

Les conteneurs ont pour classe parente **Pane** et **Region** et permettent la mise en page des composants



Taille des composants

Les noeuds que l'on peut placer dans des conteneurs possèdent des propriétés qui peuvent être prises en compte lors du calcul de leur disposition.

- **minWidth** : Largeur minimale souhaitée pour le composant
- **prefWidth** : Largeur préférée (idéale) du composant
- **maxWidth** : Largeur maximale souhaitée pour le composant
- **minHeight** : Hauteur minimale souhaitée pour le composant
- **prefHeight** : Hauteur préférée (idéale) du composant
- **maxHeight** : Hauteur maximale souhaitée pour le composant

L'effet de ces propriétés dépend du type de conteneur utilisé et de ses règles spécifiques de positionnement et de dimensionnement.

Les VBox et les HBox

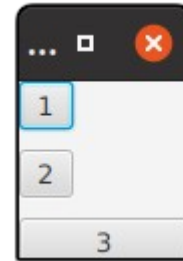
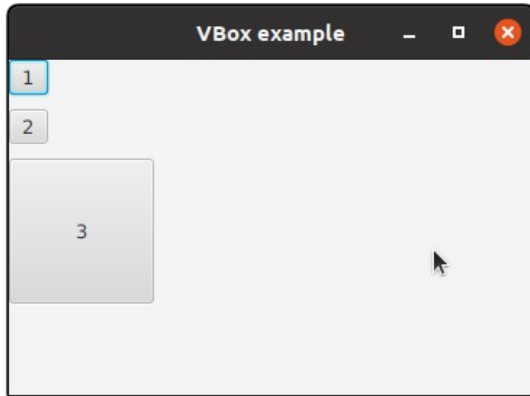
- Le conteneur **VBox** place les composants verticalement, sur une colonne. Les composants sont ainsi ajoutés à la suite les uns des autres (de haut en bas).
- Le conteneur **HBox** place les composants sur une ligne horizontale. Les composants sont ajoutés à la suite les uns des autres (de gauche à droite).
- L'ajout des composants enfants dans le conteneur s'effectue en invoquant d'abord **children** qui retourne la liste des enfants du conteneur et en y ajoutant ensuite le composant considéré (méthodes **add()** ou **addAll()**)

Dans ce qui suit, seuls les **VBox** seront présentés



Les VBox (1)

- L'alignement des composants enfants est déterminé par la propriété **alignment**, par défaut **TOP_LEFT** (type énuméré **Pos**).
- L'espacement vertical entre les composants est défini par la propriété **spacing** (0 par défaut)
- Le conteneur tant à respecter la taille préférée des composants **prefWidth**. S'il est trop petit, il les réduit jusqu'à **minWidth** et ensuite, il les affiche partiellement



Les VBox (2)

- La propriété **padding** qui est de type **Insets** permet de définir l'espace entre le bord du conteneur et les composants enfants.

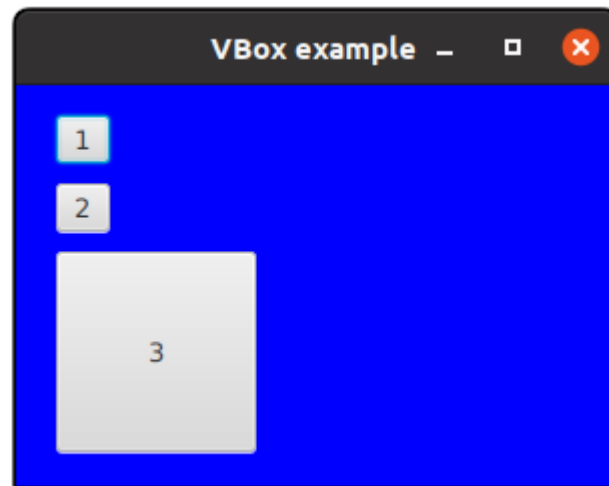
Dans le constructeur de **Insets**, les paramètres définissent les espacements dans l'ordre suivant : *Top, Right, Bottom, Left* ou une valeur identique pour les quatre côtés si l'on passe un seul paramètre

```
root.padding=Insets(15.0, 10.0, 15.0, 10.0)
```

```
root.padding=Insets(10.0)
```

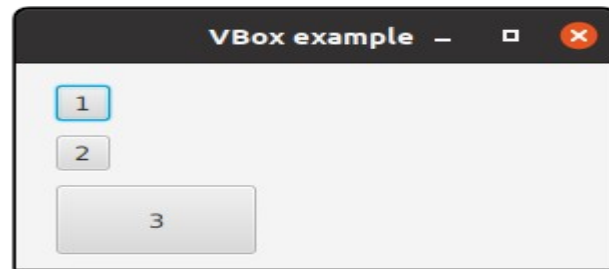
Une couleur de fond peut être appliquée au conteneur en définissant la propriété `style` qui permet de passer en chaîne de caractères, un style CSS.

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



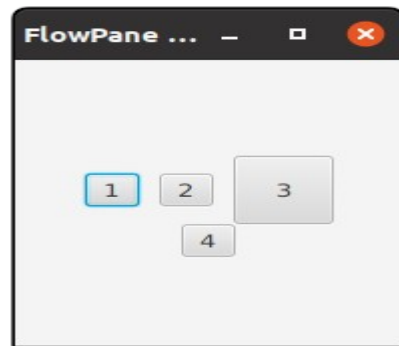
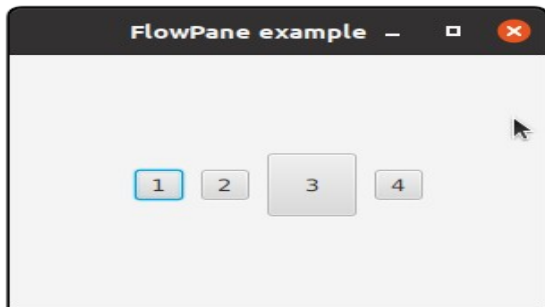
Exemple VBox

```
class ExempleVBox:Application() {  
  
    override fun start(premierStage: Stage) {  
        premierStage.title="VBox example"  
        val root = VBox()  
        // 10 pixels entre les éléments  
        root.spacing=10.0  
        // offset intérieur  
        root.padding = Insets(15.0, 20.0, 10.0, 20.0)  
        val btn1 = Button("1")  
        val btn2 = Button("2")  
        val btn3 = Button("3")  
        // préférence pour la taille  
        btn3.setPrefSize(100.0, 100.0)  
        root.children.addAll(btn1,btn2,btn3)  
        val scene = Scene(root, 300.0, 200.0)  
        premierStage.scene=scene  
        premierStage.show()  
    }  
}
```



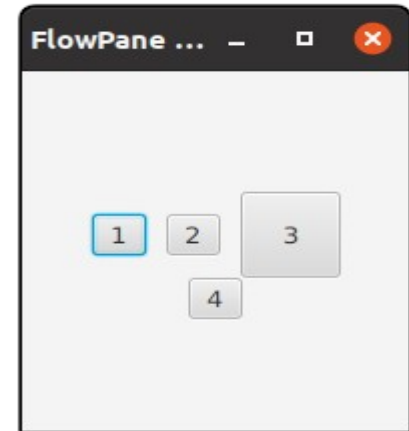
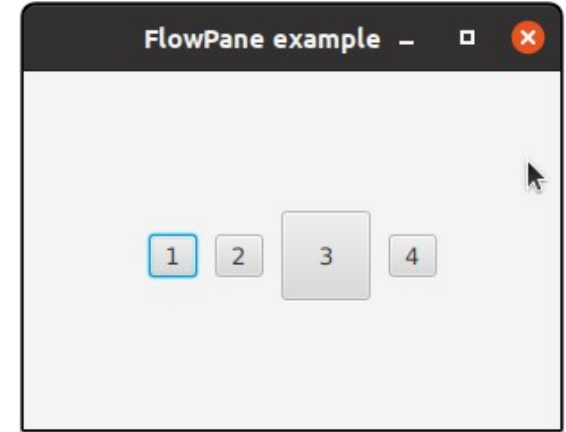
FlowPane

- Le conteneur **FlowPane** place les composants sur une ligne horizontale ou verticale et passe à la ligne ou à la colonne suivante quand il n'y a plus assez de place disponible.
- Quelques propriétés importantes du conteneur **FlowPane** :
 - **hgap (vgap)**: Espacement horizontal (vertical) entre les composants
 - **alignment**: Alignement global des composants dans le conteneur
 - **orientation**: Orientation du FlowPane (peut aussi être passé au constructeur) qui détermine s'il s'agit d'un **FlowPane** horizontal (par défaut) ou vertical.



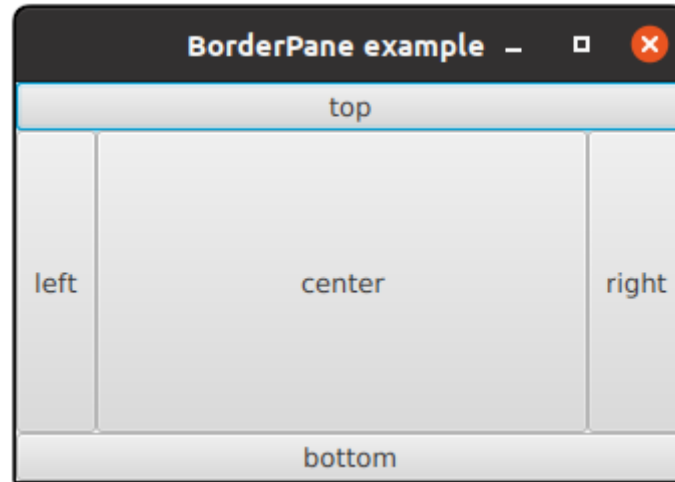
Exemple FlowPane

```
class FlowPaneHorizontal: Application() {  
    override fun start(premierStage: Stage) {  
        premierStage.title="FlowPane example"  
        val root = FlowPane(Orientation.HORIZONTAL)  
        root.hgap=10.0  
        root.alignment=Pos.CENTER  
        val btn1 = Button("1")  
        val btn2 = Button("2")  
        val btn3 = Button("3")  
        val btn4 = Button("4")  
        // préférence pour la taille  
        btn3.setPrefSize(50.0,50.0)  
        root.children.addAll(btn1,btn2,btn3, btn4)  
        val scene = Scene(root, 300.0, 200.0)  
        premierStage.scene=scene  
        premierStage.show()  
    }  
}
```



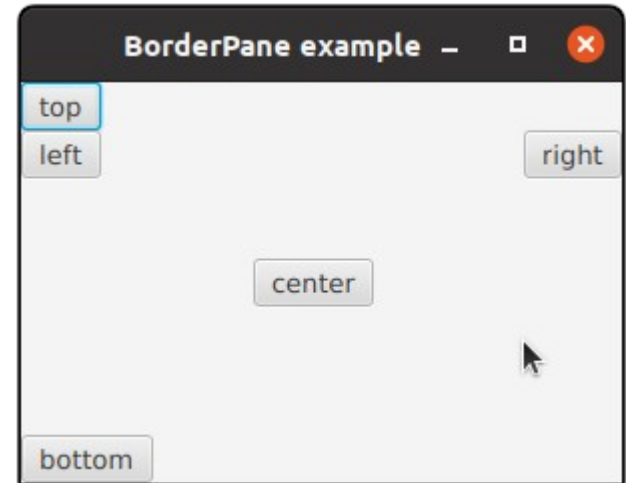
BorderPane (1)

- Le conteneur **BorderPane** permet de placer des composants dans cinq zones : top, center, left, right et bottom
- Un seul objet Node (composant, conteneur, ...) peut être placé dans chacune de ces zones.



BorderPane(2)

- Les composants placés dans les zones **top** et **bottom** gardent leur hauteur préférée. Ils sont éventuellement agrandis horizontalement jusqu'à leur largeur maximale ou réduit à leur taille minimale en fonction de la largeur du conteneur.
- Les composants placés dans les zones **left** et **right** gardent leur largeur préférée. Ils sont éventuellement agrandis verticalement jusqu'à leur hauteur maximale ou réduit à leur taille minimale en fonction de la hauteur restante entre les (éventuelles) zones **top** et **bottom** du conteneur.
- Le composant placé dans la zone **center** est éventuellement agrandi (jusqu'à sa taille maximale) ou réduit (à sa taille minimale) dans les deux directions (largeur et hauteur) pour occuper l'espace qui reste au centre du conteneur.



BorderPane(3)

On peut appliquer des contraintes de positionnement

- **setAlignment(Node, Pos)** permet de modifier l'alignement par défaut du composant passé en paramètre (point d'ancrage)
- **setMargin(Node, Insets)** fixe une marge (objet de type Insets) autour du composant passé en paramètre

```
BorderPane.setAlignment(btnTop, Pos.TOP_RIGHT)
```

```
BorderPane.setAlignment(btnRight, Pos.BOTTOM_CENTER)
```

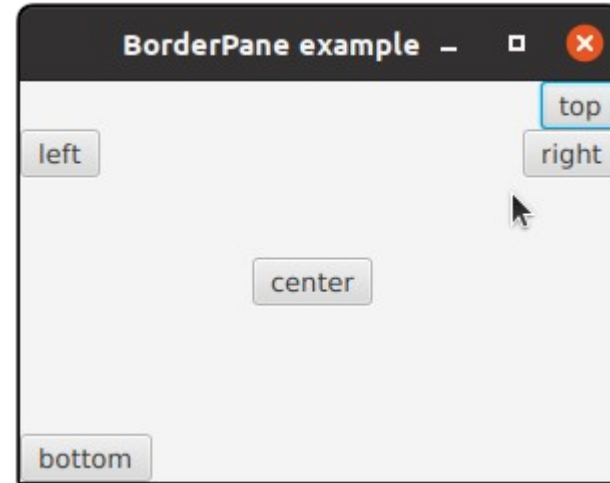
```
BorderPane.setMargin(btnBottom, Insets(10.0))
```

- Remarque : Par défaut, la taille maximale des composants est égale à leur taille préférée. Pour qu'ils s'agrandissent, il faut modifier la propriété **maxWidth** et/ou **maxHeight**

```
btn.maxWidth= Double.MAX_VALUE
```

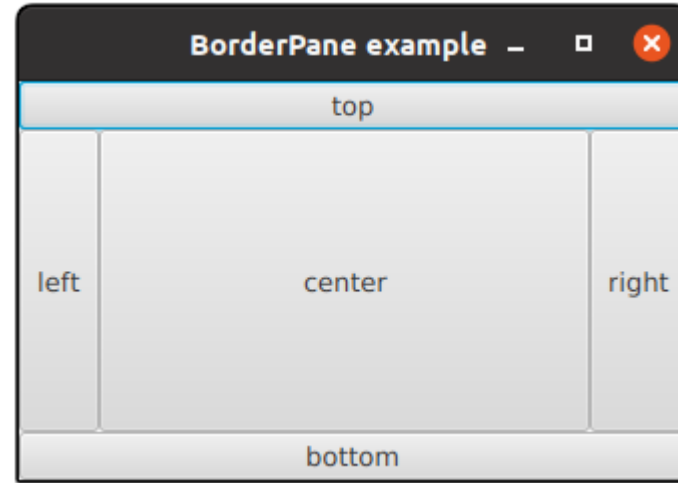
Exemple BorderLayout(1)

```
class BorderLayoutExemple: Application() {  
    override fun start(premierStage: Stage) {  
        premierStage.title = "BorderPane example"  
        val root = BorderPane()  
        val btn1=Button("top")  
        // par défaut en haut à gauche  
        root.top = btn1  
        // maintenant en haut à droite  
        BorderPane.setAlignment(btn1,Pos.TOP_RIGHT)  
        val btn2=Button("center")  
        root.center = btn2  
        val btn3=Button("left")  
        root.left = btn3  
        val btn4=Button("right")  
        root.right =btn4  
        val btn5=Button("bottom")  
        root.bottom=btn5  
        val scene = Scene(root, 300.0, 200.0)  
        premierStage.scene = scene  
        premierStage.show()  
    }  
}
```



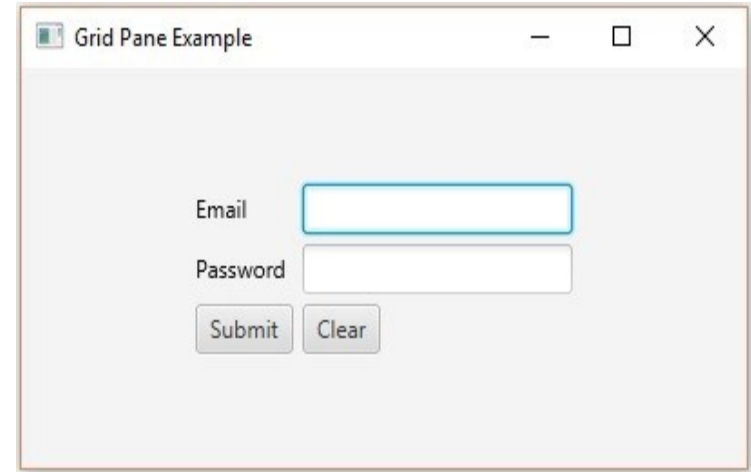
Example BorderLayout (2)

```
class BorderLayoutExample: Application() {  
    override fun start(premierStage: Stage) {  
        premierStage.title = "BorderPane example"  
        val root = BorderPane()  
        val btn1=Button("top ")  
        btn1.maxWidth=Double.MAX_VALUE  
        root.top = btn1  
        val btn2=Button("center")  
        btn2.maxWidth=Double.MAX_VALUE  
        btn2.maxHeight=Double.MAX_VALUE  
        root.center = btn2  
        val btn3=Button("left")  
        root.left = btn3  
        btn3.maxHeight=Double.MAX_VALUE  
        val btn4=Button("right")  
        root.right = btn4  
        btn4.maxHeight=Double.MAX_VALUE  
        val btn5=Button("bottom")  
        root.bottom=btn5  
        btn5.maxWidth=Double.MAX_VALUE  
        val scene = Scene(root, 300.0, 200.0)  
        premierStage.scene = scene  
        premierStage.show()  
    }  
}
```



GridPane (1)

- Le conteneur **GridPane** permet de disposer les composants dans une grille comportant des lignes et des colonnes
- La taille de la grille dépend des composants qui sont ajoutés.
- Par défaut, la hauteur (largeur) de chaque ligne (colonne) est déterminée par la hauteur (largeur) préférée du composant le plus haut (large) qui s'y trouve.



GridPane (2)

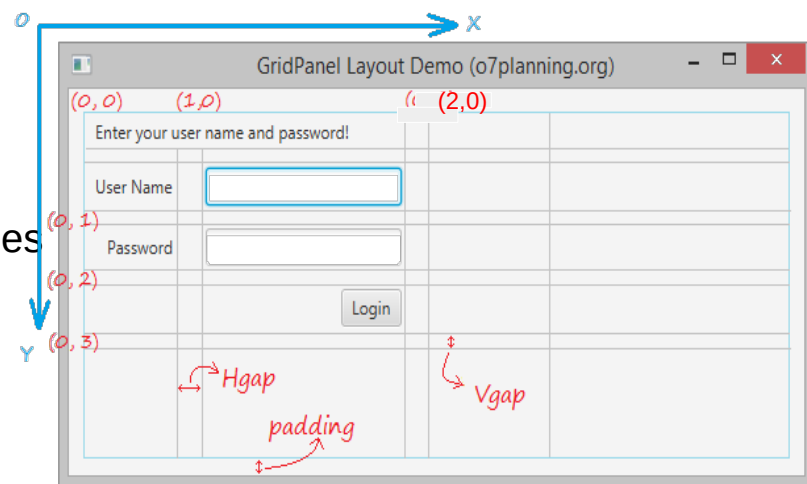
Ajout d'un composant dans la grille

- méthode **add()** qui a comme paramètre le composant ainsi que les contraintes principales de placement :
 - indice de la colonne et de la ligne (commence à 0)
 - nombre de colonnes et de lignes utilisées (par défaut: 1)

// labelUserName est placé en 0,1 dans la grille
`grille.add(labelUserName,0,1)`

- les éléments peuvent s'étendre sur plusieurs colonnes ou lignes

// labelEnter est placé en 0,0 dans la grille et s'étend sur 2 colonnes
`grille.add(labelEnter,0,0,2,1)`



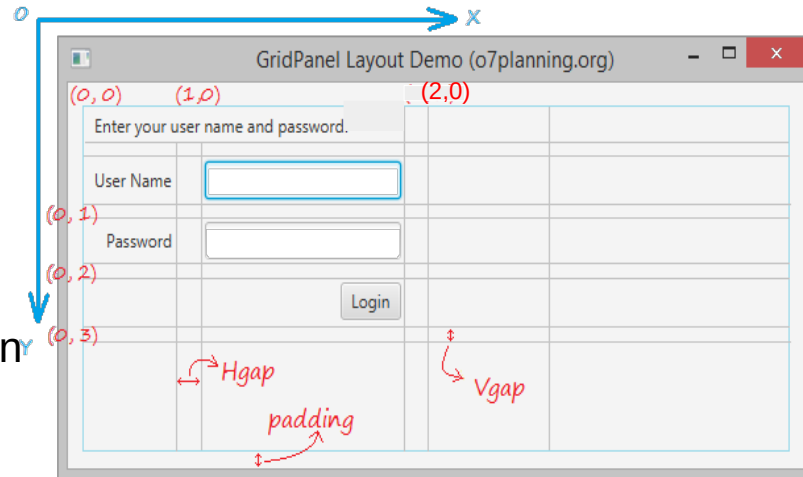
GridPane (3)

On peut connaître les coordonnées d'un composant dans la grille

```
numeroLigne=GridPane.getRowIndex(labelUserName)  
numeroColonne=GridPane.getColumnIndex(labelUserName)
```

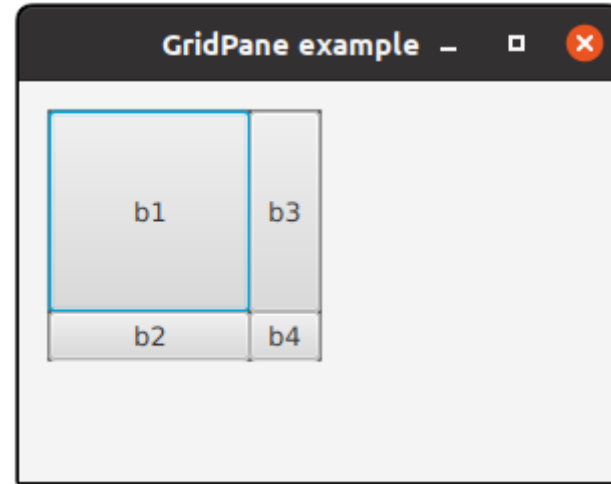
Quelques propriétés du conteneur GridPane

- **hgap** : espacement horizontal entre les colonnes
- **vgap** : espacement vertical entre les lignes
- **alignment** : alignement de la grille dans le conteneur
- **padding** : espacement autour de la grille
- **isGridLinesVisible** : affichage des lignes de construction de la grille (pour debugger, sinon utiliser du CSS)





Example GridPane

```
class GridPaneExemple: Application() {  
    override fun start(premierStage: Stage) {  
        premierStage.title = "GridPane example"  
        val root = GridPane()  
        root.isGridLinesVisible=true  
        root.padding= Insets(15.0)  
        val btn1=Button("b1")  
        btn1.setPrefSize(100.0,100.0)  
        root.add(btn1,0,0)  
        val btn2=Button("b2")  
        btn2.maxWidth=Double.MAX_VALUE  
        root.add(btn2,0,1)  
        val btn3=Button("b3")  
        root.add(btn3,1,0)  
        btn3.maxHeight=Double.MAX_VALUE  
        val btn4=Button("b4")  
        root.add(btn4,1,1)  
        val scene = Scene(root, 300.0, 200.0)  
        premierStage.scene = scene  
        premierStage.show()  
    }  
}
```



Les contraintes (1)

- La taille des lignes et des colonnes de la grille peut être gérée par des contraintes de lignes et de colonnes qui s'appliquent pour tous les composants placés dans la ligne ou la colonne concernée (**RowConstraints** et **ColumnConstraints**)
- L'ordre des ajouts des contraintes correspond à l'ordre des lignes ou des colonnes (ici contrainte liée au nombre de pixels de large)

<code>grille.columnConstraints.add(ColumnConstraints(100))</code>		contrainte sur la colonne 1
<code>grille.columnConstraints.add(ColumnConstraints(200))</code>		contrainte sur la colonne 2

- On peut aussi associer des contraintes individuelles aux composants de la grille. ces contraintes sont prioritaires sur celles appliquées globalement aux lignes et aux colonnes.

Les contraintes (2)

Les contraintes de colonnes **ColumnConstraints** possèdent les propriétés suivantes :

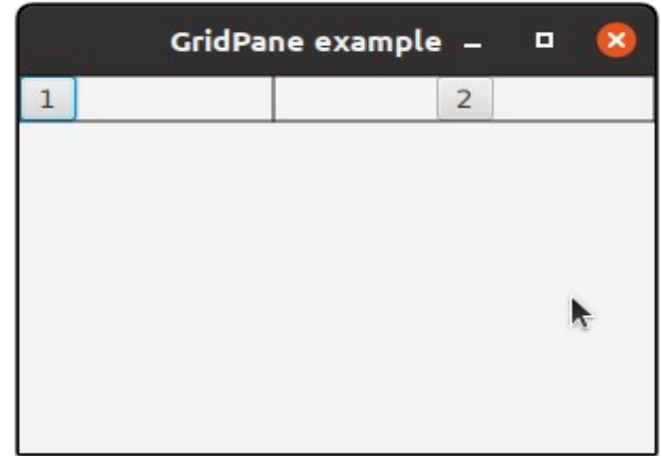
- **minWidth**: largeur minimale souhaitée pour la colonne
- **prefWidth**: largeur préférée (idéale) pour la colonne
- **maxWidth**: largeur maximale souhaitée pour la colonne
- **percentWidth**: largeur de la colonne en pourcentage de la largeur de la grille (est prioritaire sur min-, pref- et maxWidth)
- **halignment**: Alignement par défaut des composants dans la colonne (de type énuméré HPos: CENTER, LEFT, RIGHT)
- **hgrow** : Priorité d'agrandissement vertical (de type énuméré Priority : ALWAYS, SOMETIMES, NEVER)
- **fillWidth** : Booléen indiquant si le composant doit s'agrandir (true) jusqu'à sa largeur maximale ou alors garder sa largeur préférée (false). Par défaut: true

```
val contrainte = ColumnConstraints()
contrainte.minWidth=10
contrainte.maxWidth=20
contrainte.halignment=HPos.CENTER
```

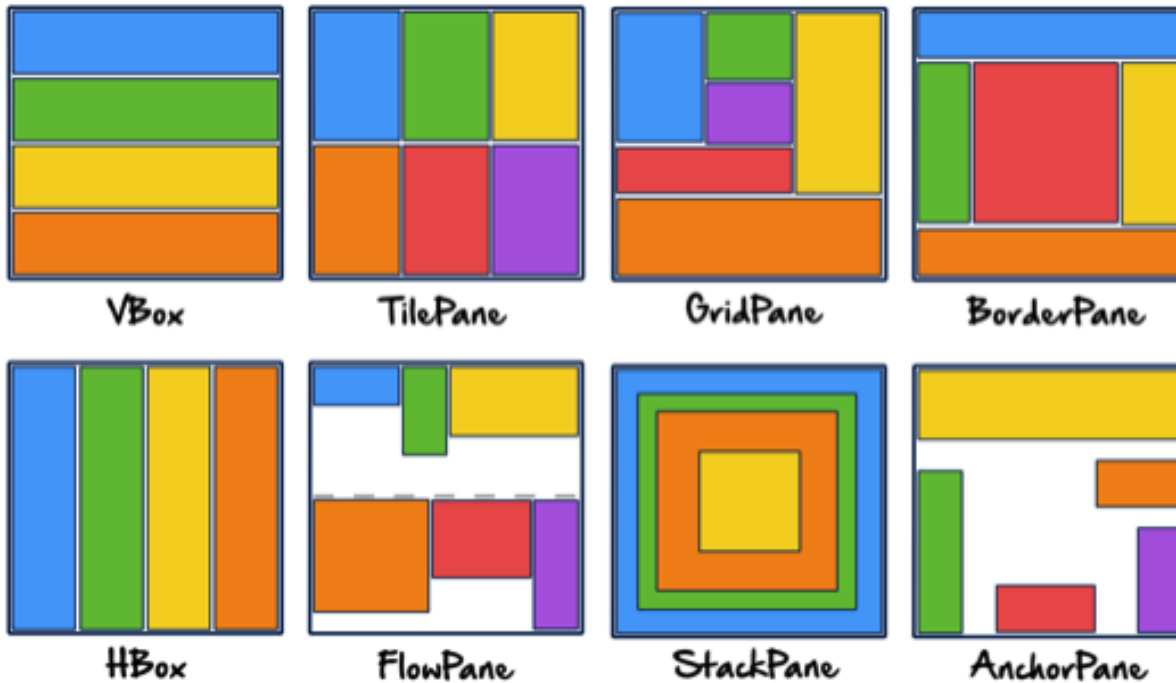
Il existe l'équivalent pour **RowConstraints**

Exemple de contraintes

```
val root = GridPane()
val scene = Scene(root, 300.0, 200.0)
primaryStage.scene = scene
root.add(Button("1"), 0, 0)
root.add(Button("2"), 1, 0)
val column1 = ColumnConstraints()
column1.percentWidth = 40.0
val column2 = ColumnConstraints()
column2.percentWidth = 60.0
column2.halignment = HPos.CENTER
root.columnConstraints.addAll(column1, column2)
root.isGridLinesVisible = true
primaryStage.show()
```

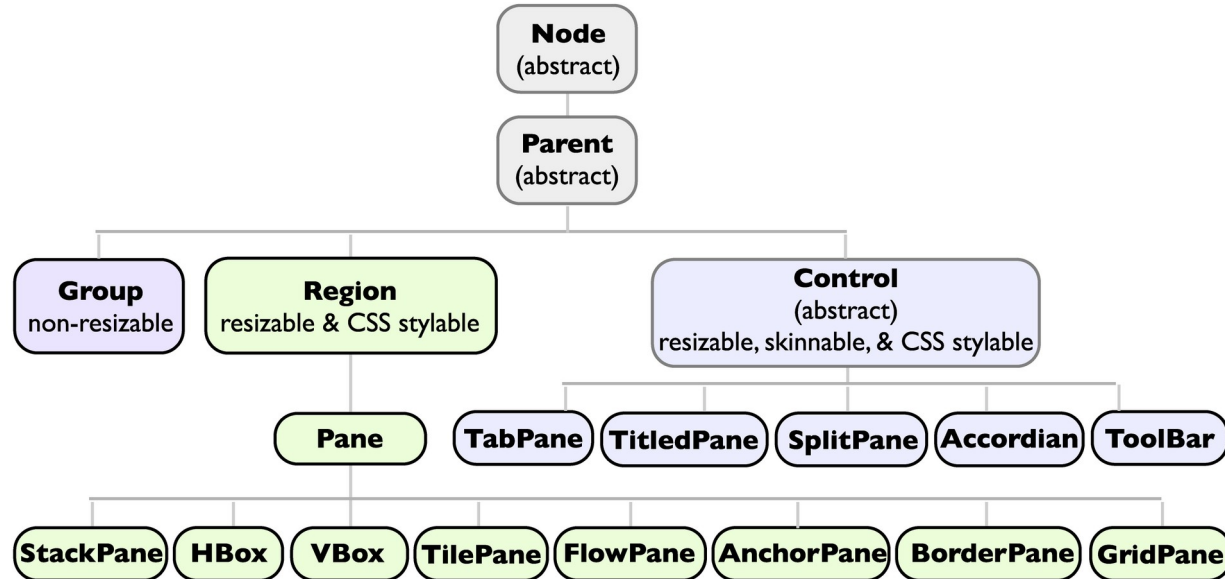


D'autres conteneurs ...

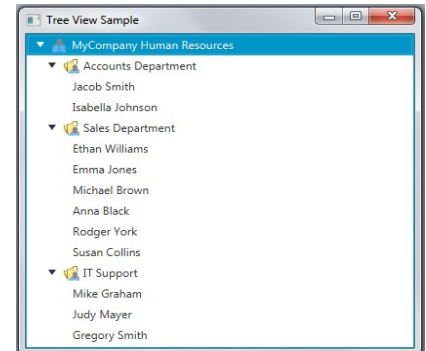
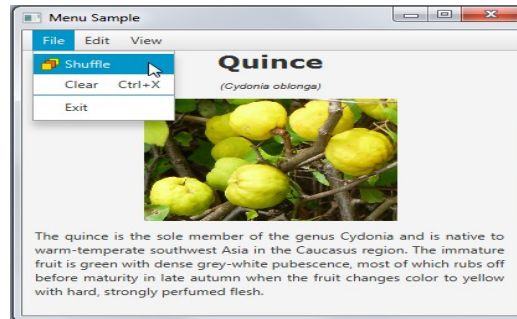
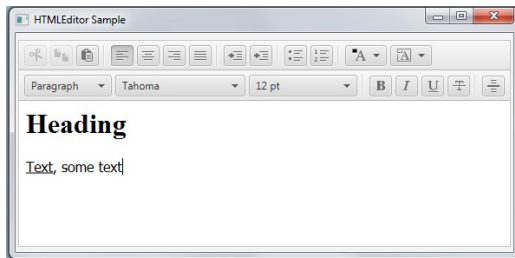
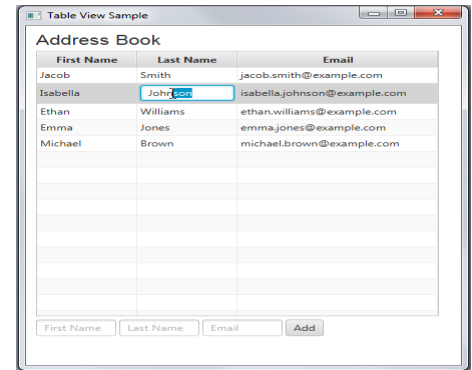
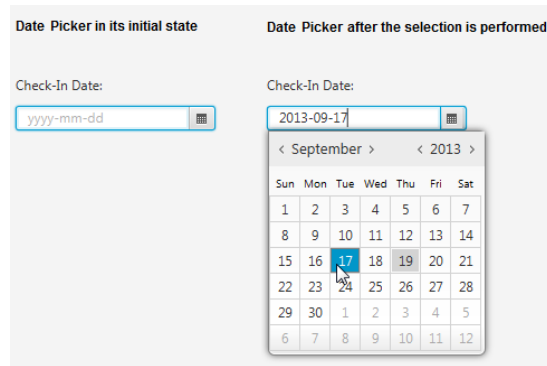


Les conteneurs

JavaFX 2.0 Layout Classes



Autres conteneurs ou Controls

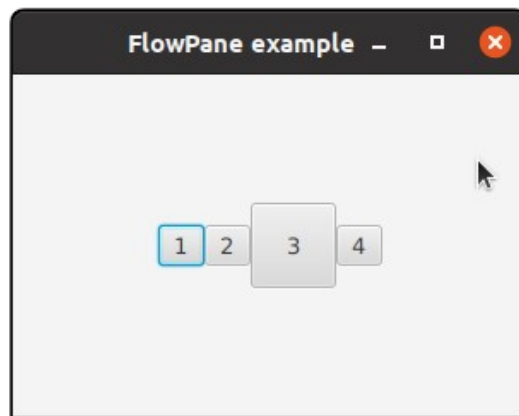


Utilisation de style CSS

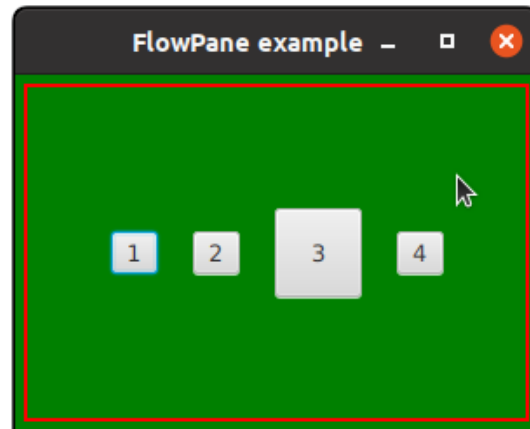
```
scene.stylesheets.add(MaClass::class.java.getResource("css/grille.css").toExternalForm())  
flowPane.setStyleClass("mygrid")
```

grille.css

```
.mygrid {  
    -fx-border-width: 2px;  
    -fx-border-color: red;  
    -fx-border-insets: 5px;  
    -fx-background-color: green;  
    -fx-hgap: 20  
}
```



Avant



Après

Quels conteneurs utiliser ?

The screenshot shows a JavaFX application window titled "TP4MVCPropre en javaFX". The interface is divided into three main sections:

- Consultation des livres:** This section displays a book titled "L'Appel de Cthulhu" in a large, bold font. Below the title, the genre "Horror" and the author "Howard Phillips Lovecraft" are listed. The text is centered between two vertical scroll bars, indicating a list of items.
- Ajout d'un auteur:** This section contains two text input fields labeled "Nom :" and "Prénom :". To the right of these fields is a vertical button with a "+" sign, used for adding a new author.
- Ajout d'un livre:** This section contains a text input field labeled "Titre :". Below it are two more input fields, likely for genre and author, each with a small downward arrow indicating a dropdown menu. To the right of these fields is a vertical button with a "+" sign, used for adding a new book.