Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2021 – 2022
ATL

**ATL – Ateliers Logiciels**

# Interfaces graphiques

*JavaFX*

### Consignes

Dans ce document vous trouverez une introduction sur l'utilisation de *JavaFX*, librairie permettant de développer des interfaces graphiques. Pour ce faire, vous pouvez télécharger les ressources du TD via le lien `https://git.esi-bru.be/ATL/javaFX/repository/archive.zip`. N'oubliez pas de vous **connecter** au serveur `git.esi-bru.be` pour avoir accès au téléchargement.

## 1 Hello World

Afin de commencer l'écriture d'une interface graphique *JavaFX* écrivons le classique `Hello World` sans nous soucier de l'ensemble des détails d'écriture. Cet exemple sera composé d'un unique champs de texte : « Hello World ».

```java
package esi.atl.fx.hello;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

import static javafx.application.Application.launch;

public class HelloWorld
      extends Application {

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

   @Override
   public void start(Stage primaryStage) {
      primaryStage.setTitle("My First JavaFX App");

      BorderPane root = new BorderPane();
      Text helloText = new Text("Hello World");
      root.setCenter(helloText);
      Scene scene = new Scene(root, 250, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```



Version du document
27 septembre 2021

De cet exemple et de la documentation [1] de la classe `Application`, nous pouvons comprendre les choses suivantes :

**JavaFX : la classe `Application`**

Une application *JavaFX* hérite de la classe `Application`. Pour lancer une telle application il faut exécuter la méthode statique `Application.launch()`.
Cette méthode :

- ▷ appelle la méthode d'initialisation `Application.init()` ;
- ▷ appelle la méthode `Application.start()`, point d'entrée de l'application, elle prend en paramètre une instance de la scène principale : `start(Stage primaryStage)` ;
- ▷ attend que l'application se termine [a] ;
- ▷ appelle la méthode `Application.stop()` qui se charge de clôturer l'application.

*a.* Nous reviendrons plus tard sur comment le système détermine la fin d'une application.

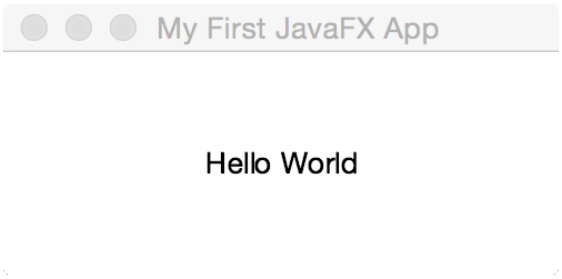


FIGURE 1 – Résultat de l'exécution du `HelloWorld`. Le *Look and Fell* dépendant de l'OS sur lequel il est exécuté.

## Exercice

Dans votre `IDE` créez le projet `MyJavaFx` avec `maven`.

Modifiez ensuite le fichier `pom.xml` afin d'ajouter :

- ▷ la librairie `javafx-controls` qui contient le code JavaFx ;
- ▷ le plugin `javafx-maven-plugin` qui permet d'exécuter une application JavaFx

Le `pom.xml` de votre projet ressemble à celui présenté ci-dessous (où $i$ est la version du JDK utilisée).

Remarquez que la balise `mainClass` du plugin définit le point d'entrée de votre programme. N'oubliez pas d'**adapter le contenu de cette balise** pour correspondre au nom de package que vous avez choisi.

Copiez le code de la classe `HelloWorld` dans ce nouveau projet et réécrivez les méthodes `init()` et `stop()` afin d'afficher un message dans la console lors de l'appel à ces méthodes. Le résultat en console de l'exécution du `HelloWorld` apparait à la figure 2 page suivante.

1. https://openjfx.io/javadoc/13/javafx.graphics/javafx/application/Application.html

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    ↪ http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>atl.sample</groupId>
   <artifactId>MyJavaFx</artifactId>
   <version>1.0</version>

   <dependencies>
      <dependency>
         <groupId>org.openjfx</groupId>
         <artifactId>javafx-controls</artifactId>
         <version>13</version>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
            <version>0.0.4</version>
            <configuration>
               <mainClass>esi.atl.fx.hello.HelloWorld</mainClass>
            </configuration>
         </plugin>
      </plugins>
   </build>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>i</maven.compiler.source>
      <maven.compiler.target>i</maven.compiler.target>
   </properties>
</project>
```

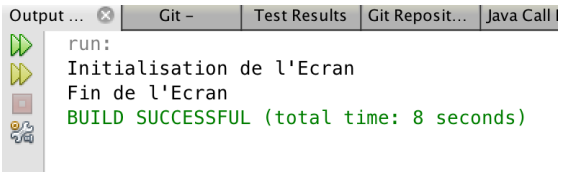


FIGURE 2 – Résultat de l'exécution du `HelloWorld` modifié. Des messages provenant de la réécriture des méthodes `init()` et `stop()` apparaissent dans la console.

**Comment exécuter mon projet :**

- en ligne de commande via `mvn clean javafx:run`
- avec `Netbeans` :
  - ouvrir la fenêtre `Navigator` via la barre de menu `Windows > Navigator` ;
  - dans la fenêtre `Navigator`, clic droit sur `javafx run` ;
  - choisir l'option `Execute Goal` ;
- avec `IntelliJ` :
  - ouvrir la fenêtre `maven` sur la barre de droite ;
  - développer les menus `JavaFX > Plugins > javafx` ;
  - clic droit sur `javafx:run` ;
  - choisir l'option `Run Maven Build` .

Oracle propose un lien [2] pour apprendre à modifier l'affichage du texte de notre interface graphique. Sur base de cette documentation essayez de changer les propriétés ci-dessous afin d'obtenir l'affichage de la figure 3 :

- la police du texte en *Times New Roman*
- la couleur du texte en rouge

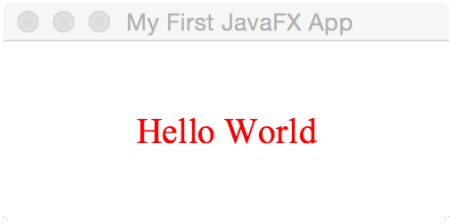


FIGURE 3 – Résultat de l'exécution du `HelloWorld` après modification des propriétés du texte.

## 2 La salle de spectacle

La description d'une application JavaFX suit l'analogie de la salle de spectacle [3]. On y trouve une `Stage` qui constitue l'endroit où se déroule le spectacle [4].

```
import javafx.stage.Stage
```

Ensuite on trouve la `Scene`, endroit où se déroule l'action.

```
import javafx.scene.Scene
```

Au sein de cette `Scene` on trouve un premier `Layout` dont le but est d'organiser l'interface graphique. Nous étudierons ceux-ci en détail à la section 4 page 13.

```
import javafx.scene.layout.BorderPane
```

Le `Layout BorderPane` divise l'écran en cinq zones : *Top*, *Bottom*, *Left*, *Right* et *Center*.

2. http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/text-settings.htm
3. theater en anglais
4. ce qui se traduirait par Scène en français. Malheureusement le terme anglais Scene sera utilisé par la suite, ce qui pourrait créer des confusions

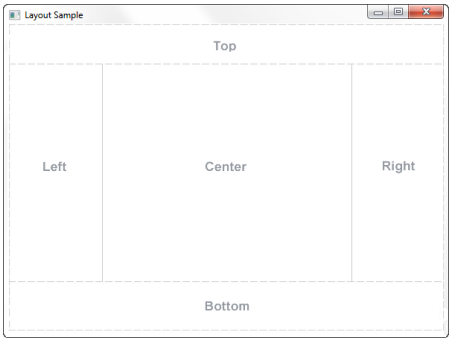


FIGURE 4 – Le *layout* `BorderPane` divise un écran en 5 zones

Finalement dans une des zones du `Layout` nous trouvons les composants de l'interface graphique. Il s'agit des libellés, boutons ou autres *CheckBox* constituant les éléments actifs de l'application. Ils sont appelés `Controls`. Chaque composant aura droit à une importation spécifique, dans notre exemple il s'agit de l'import suivant :

```
import javafx.scene.text.Text
```

### Question 1

1. Dans la classe `HelloWorld` modifiez le code d'instanciation de la scène ci-dessous en modifiant la valeur des paramètres numériques.

   ```
   Scene scene = new Scene(root, 250, 100);
   ```

   Quel est l'effet de ce changement de paramètres ?

2. Juste après avoir modifié le titre de la `primaryStage` ajoutez le code ci-dessous :

   ```
   primaryStage.initStyle(StageStyle.TRANSPARENT);
   ```

   Quel est l'effet de cette méthode `initStyle()` sur la fenêtre ? Après consultation de la javadoc [5] de la classe `Stage` donnez les valeurs possibles de l'énumération `StageStyle`.

3. Afin de placer le composant `Text` au centre de l'écran, la méthode `setCenter()` de la classe `BorderPane` est appelée

   ```
   BorderPane root = new BorderPane();
   root.setCenter(helloText);
   ```

   Que ce passe-t-il si on choisit d'appeler une des méthodes ci-dessous ?

   - ▷ `setTop()`
   - ▷ `setBottom()`
   - ▷ `setLeft()`
   - ▷ `setRight()`

## 3 Les composants, controls, widgets

Une interface graphique passe par l'utilisation de boutons, de libellés, de zone de texte, de cases à cocher et de différents objets appelés *composants* [6].

5. https://openjfx.io/javadoc/13/javafx.graphics/javafx/stage/Stage.html
6. Ces composants d'interface sont fréquemment nommés *controls* dans la documentation en anglais

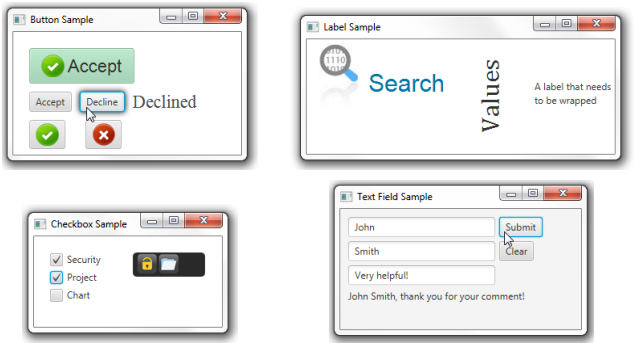


TABLE 1 – Différents composants JavaFX : Bouton, Label, Check Box et Text Field

Vous trouverez des tutoriels sur les différents composants à l'adresse suivante [7] .

Le but des prochaines sections est de vous familiariser avec certains composants et de vous inviter à lire la documentation qui les concernent. Pour comprendre le fonctionnement des composants non explorés dans ce document, il faudra vous baser uniquement sur la documentation. Notez que la liste des composants s'enrichit chaque jour, notamment par des projets tel que ControlsFX [8] .

## 3.1 Labeled

De nombreux composants affichent et gèrent des textes (libellés, boutons, cases à cocher, etc). Afin de structurer au mieux le code, les différents composants gérant du texte vont hériter de la classe `Labeled`, comme vous pouvez le voir sur la figure 5.

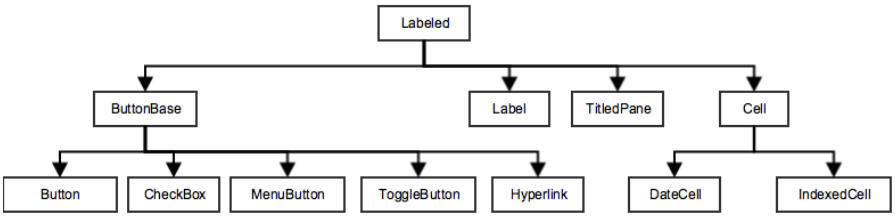


FIGURE 5 – Les composants héritant de `Labeled` peuvent gérer du texte. L'ensemble des composants bénéficiant de cet héritage ne sont pas repris sur la figure.

**Propriétés**

Afin de décrire un composant on lui associe un ensemble de *propriétés*. Celles-ci sont des attributs de notre composant :

- ▷ on accède aux propriétés d'un composant via un getter ;
- ▷ on modifie les propriétés d'un composant via un setter.

Dans le cas d'un composant `Labeled`, voici trois propriétés intéressantes :

- ▷ le texte affiché : `text` ;
- ▷ la police : `font` ;
- ▷ la couleur du texte : `textFill`.

7. http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm
8. http://fxexperience.com/controlsfx/features/

Pour obtenir la liste des propriétés d'un composant `Labeled`, vous pouvez consulter la javadoc du composant [9] .

### 3.1.1 Label

Le composant `Label` représente un texte non éditable, un libellé. Ce composant hérite de `Labeled` et de ses propriétés. Modifions notre `HelloWorld` pour utiliser ces propriétés.

```
1  package esi.atl.fx.hello;
2
3  //import ...
4
5  public class HelloWorldProperty
6        extends Application {
7
8     public static void main(String[] args) {
9        launch(args);
10    }
11
12    @Override
13    public void start(Stage primaryStage) {
14       primaryStage.setTitle("My First JavaFX App");
15       BorderPane root = new BorderPane();
16       Label helloText = new Label("Hello World");
17       helloText.setTextFill(Color.RED);
18       helloText.setFont(Font.font("Verdana", 20));
19
20       System.out.println("Le message du Libellé est " + helloText.getText());
21       System.out.println("La police du Libellé est " + helloText.getFont());
22       System.out.println("La couleur du Libellé est " + helloText.getTextFill());
23
24       root.setCenter(helloText);
25       Scene scene = new Scene(root, 250, 100);
26       primaryStage.setScene(scene);
27       primaryStage.show();
28    }
29
30 }
```

Comme vous pouvez le voir dans la documentation [10] plusieurs constructeurs sont disponibles pour le composant `Label`, dans notre exemple nous utilisons celui qui prend en paramètre le texte à afficher à l'écran. Remarquez également la manière d'accéder à la propriété `TextFill` du composant `helloText`.

## Exercice

En vous aidant de la documentation modifiez la classe `HelloWorldProperty` pour que le texte soit souligné.

### 3.1.2 Check Box

Ce composant est typiquement utilisé lorsque l'utilisateur peut choisir parmi plusieurs options qui peuvent être simultanément activées. Le composant `CheckBox` représente une case à cocher qui est caractérisée par trois états : Désélectionné, Sélectionné, Indéterminé. Chaque clic de l'utilisateur fera passer le composant d'un état à un autre.

Comme expliqué dans la javadoc [11] cette classe dispose de deux constructeurs :

9. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/Labeled.html
10. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/Label.html
11. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/CheckBox.html

- `new CheckBox()` : une `CheckBox` sans libellé ;
- `new CheckBox("Faites votre choix")` : une `CheckBox` avec libellé.

Les propriétés non héritées d'une `CheckBox` sont :

- `allowIndeterminate` : si l'attribut prend la valeur `false`, la check box passe par deux états (sélectionné, déselectionné), dans le cas contraire on ajoute un troisième état (indeterminé) ;
- `indeterminate` : indique si l'état de la case est indeterminé ;
- `selected` : indique si la case est sélectionnée.

code/HelloWorldCheckBox.java

```
1  package esi.atl.fx.hello;
2  //import ...
3
4  public class HelloWorldCheckBox
5        extends Application {
6
7     public static void main(String[] args) {
8        launch(args);
9     }
10
11    @Override
12    public void start(Stage primaryStage) {
13       primaryStage.setTitle("My First JavaFX App");
14       BorderPane root = new BorderPane();
15
16       CheckBox checkBox1 = new CheckBox();
17       checkBox1.setText("First");
18       checkBox1.setSelected(true);
19
20       CheckBox checkBox2 = new CheckBox("Second");
21       checkBox2.setIndeterminate(true);
22
23       CheckBox checkBox3 = new CheckBox("Third");
24
25       checkBox3.setAllowIndeterminate(true);
26
27       //Alignment
28       root.setLeft(checkBox1);
29       BorderPane.setAlignment(checkBox1, Pos.CENTER);
30       root.setCenter(checkBox2);
31       root.setRight(checkBox3);
32       BorderPane.setAlignment(checkBox3, Pos.CENTER);
33
34       Scene scene = new Scene(root, 250, 100);
35       primaryStage.setScene(scene);
36       primaryStage.show();
37    }
38 }
```

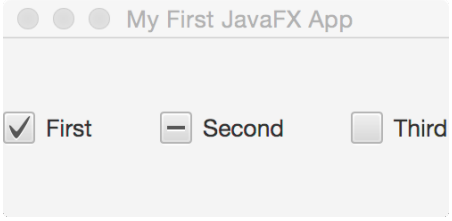


FIGURE 6 – Résultat de l'exécution du `HelloWorldCheckBox`. Chaque box est dans un état différent

### Question 2

Sur base du code `HelloWorldCheckBox` ci-avant.

1. Quel est la différence de comportement entre les trois `CheckBox` de la classe `HelloWorldCheckBox` ?
2. Quel est l'effet sur l'affichage des `CheckBox` si on supprime les lignes de code ci-dessous ?

```
BorderPane.setAlignment(checkBox1, Pos.CENTER);
BorderPane.setAlignment(checkBox3, Pos.CENTER);
```

## 3.2 TextInputControl

Comme on le voit sur la figure 7 la classe abstraite `TextInputControl`[12] est la classe parente de différents composants qui permettent à l'utilisateur de saisir du texte. Il s'agit notamment des composants d'interface tels que `TextField`, `PasswordField` et `TextArea`.

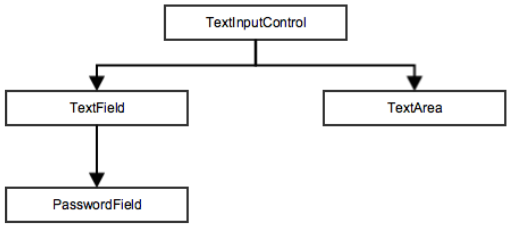


FIGURE 7 – La classe abstraite `TextInputControl` est la classe parente des composants `TextField`, `PasswordField` et `TextArea`.

Cette classe définit les propriétés de base et les fonctionnalités communes aux composants offrant une saisie de texte. Ces propriétés permettent la sélection et l'édition de texte ou la gestion du curseur à l'intérieur du texte (*caret*). On peut citer les propriétés :

- ▷ `text` : le texte contenu dans le composant ;
- ▷ `editable` : booléen rendant le texte éditable par l'utilisateur ;
- ▷ `font` : la police du texte ;
- ▷ `length` : la longueur du texte.

Ainsi que les méthodes :

- ▷ `clear()` : efface le texte du composant ;
- ▷ `insertText()` : insère une chaine de caractères dans le texte ;
- ▷ `appendText()` : ajoute une chaine de caractères à la fin du texte ;
- ▷ `selectAll()` : sélectionne l'ensemble du texte.

### 3.2.1 TextField

Le composant `TextField`[13] représente un champ texte d'une seule ligne qui est éditable par défaut. Il peut également être utilisé pour afficher du texte. Ce composant hérite des propriétés de `TextInputControl` mais possède également les propriétés suivantes :

12. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/TextInputControl.html
13. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/TextField.html

- `alignment` : défini l'alignement du texte ;
- `prefColumnCount` : définit le nombre de colonnes pour ce champs texte. La valeur par défaut est de 12 ;
- `onAction` : définit un événement à générer lors d'une certaine action de l'utilisateur, par défaut, lorsque l'utilisateur presse la touche *enter*.

Un exemple d'utilisation des deux premières propriétés est décrit dans l'exemple ci-dessous :

```java
package esi.atl.fx.hello;
//import ...
public class HelloWorldTextField extends Application {

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

   @Override
   public void start(Stage primaryStage) {
      primaryStage.setTitle("My First JavaFX App");
      BorderPane root = new BorderPane();

      Label userName = new Label("User Name");

      TextField tfdUserName = new TextField();
      tfdUserName.setPrefColumnCount(12);
      tfdUserName.setAlignment(Pos.CENTER_LEFT);

      //Alignment
      root.setTop(userName);
      BorderPane.setAlignment(userName, Pos.CENTER);
      root.setCenter(tfdUserName);

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

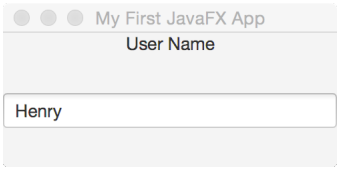


FIGURE 8 – Résultat de l'exécution du `HelloWorldtextField`. Le texte est aligné sur la gauche.

## Question 3

1. Modifiez la classe `HelloWorldtextField` et transformez le composant `TextField` en un `PasswordField`. Quelles sont les conséquences de cette modification ?
2. Lors de l'exécution de `HelloWorldTextFieldAction`, que se passe-t-il après avoir pressé la touche *enter* ?

```java
package esi.atl.fx.hello;
//import ...

public class HelloWorldTextFieldAction extends Application {

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

   @Override
   public void start(Stage primaryStage) {
      primaryStage.setTitle("My First JavaFX App");
      BorderPane root = new BorderPane();

      Label userName = new Label("User Name");
      Label test = new Label("User name saved! You can't change it");

      TextField tfdUserName = new TextField();
      tfdUserName.setPrefColumnCount(12);
      tfdUserName.setAlignment(Pos.CENTER_LEFT);
      tfdUserName.setOnAction(new EventHandler<ActionEvent>() {
         @Override
         public void handle(ActionEvent event) {
            root.setBottom(test);
            BorderPane.setAlignment(test, Pos.CENTER);
            tfdUserName.setEditable(false);
            tfdUserName.setAlignment(Pos.CENTER);
         }
      });

      //Alignment
      root.setTop(userName);
      BorderPane.setAlignment(userName, Pos.CENTER);
      root.setCenter(tfdUserName);

      Scene scene = new Scene(root, 300, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```

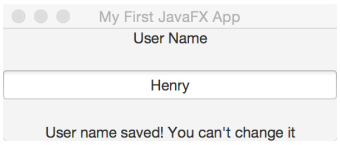


FIGURE 9 – Résultat de l'exécution du `HelloWorldTextFieldAction`. Après avoir écrit votre nom, pressez la touche *enter*.

### 3.2.2 TextArea

Le composant `TextArea`[14] permet d'afficher et de saisir du texte dans un champ multilignes (une zone de texte). Le texte peut être renvoyé à la ligne automatiquement (wrapping) et des barres de défilement (scrollbar) horizontales et/ou verticales sont ajoutées automatiquement si la taille du composant ne permet pas d'afficher l'entiereté du texte. Tous les caractères du texte possèdent les mêmes attributs (police, style, taille, couleur, ...).

14. https://openjfx.io/javadoc/13/javafx.controls/javafx/scene/control/TextArea.html

## Question 4

Dans le code de présentation ci-dessous, nous avons ajouté l'utilisation du composant `Button` qui possède également la propriété `onAction`. Essayez de comprendre ce qui se produit lors d'une pression sur le bouton de cet écran et expliquez-le brièvement.

```
1  package esi.atl.fx.hello;
2
3  //import ...
4
5  public class HelloWorldTextArea
6        extends Application {
7
8     public static void main(String[] args) {
9        launch(args);
10    }
11
12    @Override
13    public void start(Stage primaryStage) {
14       primaryStage.setTitle("My First JavaFX App");
15       BorderPane root = new BorderPane();
16
17       TextArea txaUserName = new TextArea();
18       txaUserName.setPrefColumnCount(12);
19       txaUserName.setPrefRowCount(5);
20       txaUserName.setWrapText(true);
21
22       Button btnPrint = new Button("Print");
23       btnPrint.setOnAction(new EventHandler<ActionEvent>() {
24
25          @Override
26          public void handle(ActionEvent event) {
27             System.out.println(txaUserName.getText());
28          }
29       });
30
31       //Alignment
32       root.setTop(txaUserName);
33       BorderPane.setAlignment(txaUserName, Pos.CENTER);
34       root.setCenter(btnPrint);
35
36       Scene scene = new Scene(root, 250, 300);
37       primaryStage.setScene(scene);
38       primaryStage.show();
39    }
40 }
```


FIGURE 10 – Résultat de l'exécution du `HelloWorldTextArea`. Après avoir écrit votre texte dans la zone prévue à cet effet, appuyez sur le bouton *Print*.

# 4 Les Layouts

Jusqu'à présent le positionnement des composants au sein d'un écran s'est fait en utilisant les méthodes `setCenter()` ou `setAlignment()` d'un objet `BorderPane`. Celui-ci fait partie de la famille des `layouts` [15]. Ces layouts permettent de définir la position au sein d'une fenêtre des différents composants, les positions relatives de chaque composant ainsi que les alignements et espacements de ceux-ci.

À moins que ce ne soit spécifié explicitement par le programmeur la disposition des composants est déléguée à des gestionnaires de disposition (*layout managers*) qui sont associés à ces conteneurs.

## 4.1 Le graphe de scène

Le graphe de scène (*scene graph*) est une notion importante qui représente la structure hiérarchique de l'interface graphique.

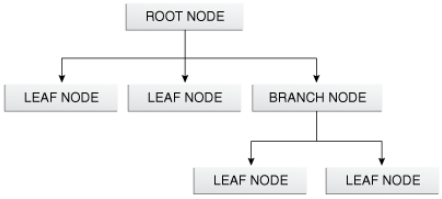


FIGURE 11 – Représentation d'un graphe de scène

Il s'agit d'un arbre orienté constitué d'un noeud origine (root) et de noeuds enfants. Tous les éléments contenus dans un graphe de scène sont des objets qui ont pour classe parente la classe `Node`. Les *layouts* et les composants héritent tous de cette classe. Cette représentation en terme d'arbre justifie l'utilisation de la méthode `node.getChildren()` avant l'ajout d'un composant à un *layout*.

### Question 5

Vérifiez via la documentation ce que retourne la méthode `getChildren()`. Quel peut être l'intérêt d'un tel type de retour ?

## 4.2 Les différents Layouts

### 4.2.1 BorderPane

Le conteneur `BorderPane` est divisé en cinq zones qui peuvent chacune contenir un seul objet `Node` : *Top*, *Bottom*, *Left*, *Right* et *Center*.

il est à noter que

- Les composants placés dans les zones *Top* et *Bottom* conservent leurs hauteurs préférées (`prefHeightProperty`) et sont éventuellement agrandis ou réduits horizontalement en fonction de la largeur du conteneur ;
- De la même façon les composants placés dans les zones *Left* et *Right* conservent leurs largeurs préférées (`prefWidthProperty`) et sont éventuellement agrandis ou réduits verticalement ;
- Le composant placé dans la zone *Center* est éventuellement redimensionné pour occuper l'espace restant au centre du conteneur ;

15. Appelés également panes, layout-panes ou conteneurs

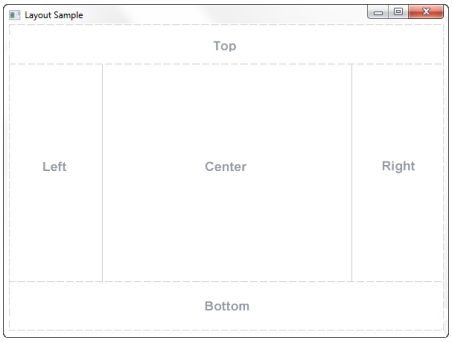


FIGURE 12 – Le *layout* `BorderPane` divise un écran en 5 zones

▷ Si aucun composant n'est ajouté à une zone, celle-ci n'occupe plus aucun espace.

N'oubliez pas que pour gérer la position d'un noeud au sein d'une zone, les méthodes suivantes sont disponibles :

▷ `alignment()` : permet de modifier l'alignement par défaut du composant passé en paramètre ;

▷ `margin()` : fixe une marge (objet de type `Insets`) autour du composant passé en paramètre.

### 4.2.2 HBox, VBox

Le *layout* HBox [16] place les composants sur une ligne horizontale de gauche à droite à la suite les uns des autres. La classe `HBoxSample` propose un exemple de répartition de trois composants `CheckBox`. Remarquez l'utilisation de la méthode `getChildren()` afin d'ajouter les composants au *layout*.

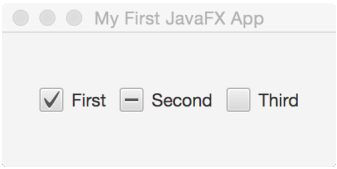


FIGURE 13 – Résultat de l'exécution du `HBoxSample`.

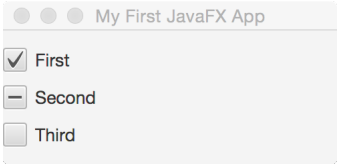


FIGURE 14 – Résultat de l'exécution du `VBoxSample`.

16. https://openjfx.io/javadoc/13/javafx.graphics/javafx/scene/layout/HBox.html

```java
package esi.atl.fx.layout;

//import ...

public class HBoxSample extends Application {

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

   @Override
   public void start(Stage primaryStage) throws Exception {
      primaryStage.setTitle("My First JavaFX App");
      HBox root = new HBox(10);
      root.setAlignment(Pos.CENTER);

      CheckBox checkBox1 = new CheckBox();
      checkBox1.setText("First");
      checkBox1.setSelected(true);

      CheckBox checkBox2 = new CheckBox("Second");
      checkBox2.setIndeterminate(true);

      CheckBox checkBox3 = new CheckBox("Third");

      checkBox3.setAllowIndeterminate(true);

      root.getChildren().add(checkBox1);
      root.getChildren().add(checkBox2);
      root.getChildren().add(checkBox3);

      Scene scene = new Scene(root, 250, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }

}
```

## Question 6

1. Le *layout* `VBox`[17] place les composants sur une ligne verticale de haut en bas les uns en dessous des autres. Transformez la classe `HBoxSample` en une classe `VBoxSample` afin de disposer d'un écran comme sur la figure 14 page précédente ;
2. Pour ajouter les composants au *layout* vous avez utilisé la méthode `addChildren()`. Pouvez-vous modifier votre code en utilisant la méthode `addAll()` ?
3. Dans la documentation du `Layout VBox` vous trouverez la remarque suivante : *VBox does not clip its content by default, so it is possible that childrens' bounds may extend outside its own bounds if a child's min size prevents it from being fit within the vbox.*. Quel impact sur le développement de nos interfaces graphiques peut avoir cette notion de *clip* ? Quelles méthodes sont à notre disposition pour gérer cette notion ?

### 4.2.3 GridPane

Le conteneur `GridPane`[18] permet de disposer les différents composants au sein d'une grille. Cette grille peut être irrégulière, les dimensions de ses cases ne sont pas nécessairement uniformes. La zone occupée par un composant peut s'étendre (*span*) sur plusieurs

17. https://openjfx.io/javadoc/13/javafx.graphics/javafx/scene/layout/VBox.html
18. https://openjfx.io/javadoc/13/javafx.graphics/javafx/scene/layout/GridPane.html

lignes et/ou colonnes. Lors de la construction d'un `GridPane` il est inutile de spécifier les nombres de lignes et de colonnes attendus, ceux-ci sont déterminés par les endroits où sont placés les composants. De même, la taille des cases de la grille est déterminée par la taille des composants les plus imposants au sein d'une ligne ou d'une colonne. Comme le montre la classe `GridPaneSample` la méthode `add()` d'un `GridPane` prend en paramètres la position de chaque composant.

```java
package esi.atl.fx.layout;

//import ...

public class GridPaneSample extends Application {

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

   @Override
   public void start(Stage primaryStage) throws Exception {
      primaryStage.setTitle("My First JavaFX App");
      GridPane root = new GridPane();
      root.setPadding(new Insets(10));
      root.setHgap(10);
      root.setVgap(5);

      Label lblTitle = new Label("JavaFX Course Login");
      lblTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
      lblTitle.setTextFill(Color.RED);
      root.add(lblTitle, 0, 0, 2, 1);
      GridPane.setHalignment(lblTitle, HPos.CENTER);
      GridPane.setMargin(lblTitle, new Insets(0, 0, 10, 0));

      Label lblUserName = new Label("User Name or email");
      GridPane.setHalignment(lblUserName, HPos.RIGHT);
      root.add(lblUserName, 0, 1);

      TextField tfdUserName = new TextField();
      tfdUserName.setPrefColumnCount(20);
      root.add(tfdUserName, 1, 1);

      Label lblPassword = new Label("Password");
      root.add(lblPassword, 0, 2);

      TextField tfdPassword = new TextField();
      tfdPassword.setPrefColumnCount(12);
      root.add(tfdPassword, 1, 2);

      GridPane.setHalignment(lblPassword, HPos.RIGHT);
      GridPane.setFillWidth(tfdPassword, false);
      Scene scene = new Scene(root);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```

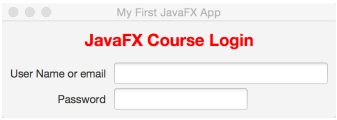


FIGURE 15 – Résultat de l'exécution du `GridPaneSample`

### Question 7

1. Au sein d'un `GridPane` placez plusieurs composants dans une même cellule. Comment sont-ils répartis au sein de cette cellule ?
2. Remplacez les paramètres de la méthode statique `GridPane.setHalignment( lblPassword, HPos.RIGHT)` par `GridPane.setHalignment(lblPassword, HPos.CENTER)`. Quel est l'impact sur l'affichage de l'écran ?
3. Remplacez les paramètres de la méthode statique `GridPane.setFillWidth( tfdPassword, false)` par `GridPane.setFillWidth(tfdPassword, true)`. Quel(s) changements(s) pouvez-vous noter ?

#### 4.2.4 Imbrication de *Layouts* : GridPane et Hbox

Les alignements proposés par l'utilisation d'un unique *layout* ne suffisent pas toujours pour réaliser l'écran imaginé par le développeur. Comme le montre la classe `MixSample` (ci-après) et la figure 16, il est très fréquent d'imbriquer plusieurs conteneurs pour obtenir la disposition désirée des composants de l'interface.


FIGURE 16 – Résultat de l'exécution du `MixSample`

## 4.3 Autres *Layouts* disponibles

Au delà des différents *layouts* présentés dans ce document, on trouve dans la librairie *JavaFX* d'autre *layouts* :

- `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 ;
- `StackPane` : empile les composants enfants les uns au dessus des autres dans l'ordre d'insertion ;
- `TilePane` : place les composants dans une grille alimentée soit horizontalement (par ligne, de gauche à droite) soit verticalement (par colonne, de haut en bas) ;
- `AnchorPane` : permet de positionner (ancrer) les composants enfants à une certaine distance des cotés du conteneur.

Des exemples d'utilisation de tous ces *layouts* sont disponibles sur le tutoriel [19] oracle.

19. http://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm

code/MixSample.java

```java
package esi.atl.fx.layout;

//import ...

public class MixSample
      extends Application {

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

   @Override
   public void start(Stage primaryStage) throws Exception {
      primaryStage.setTitle("My First JavaFX App");
      primaryStage.setMinWidth(300);
      primaryStage.setMinHeight(200);
      GridPane root = new GridPane();

      root.setAlignment(Pos.CENTER);
      root.setPadding(new Insets(20));
      root.setHgap(10);
      root.setVgap(15);

      Label lblTitle = new Label("JavaFX Course Login");
      lblTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
      lblTitle.setTextFill(Color.RED);
      root.add(lblTitle, 0, 0, 2, 1);
      GridPane.setHalignment(lblTitle, HPos.CENTER);
      GridPane.setMargin(lblTitle, new Insets(0, 0, 10, 0));

      Label lblUserName = new Label("User Name or email");
      GridPane.setHalignment(lblUserName, HPos.RIGHT);
      root.add(lblUserName, 0, 1);

      TextField tfdUserName = new TextField();
      tfdUserName.setPrefColumnCount(20);
      root.add(tfdUserName, 1, 1);

      Label lblPassword = new Label("Password");
      root.add(lblPassword, 0, 2);

      PasswordField pwfPassword = new PasswordField();
      pwfPassword.setPrefColumnCount(12);
      root.add(pwfPassword, 1, 2);

      GridPane.setHalignment(lblPassword, HPos.RIGHT);
      GridPane.setFillWidth(pwfPassword, false);

      HBox btnPanel = new HBox(12);

      Button btnLogin = new Button("Login");
      Button btnCancel = new Button("Cancel");

      btnPanel.getChildren().addAll(btnLogin,btnCancel);
      btnPanel.setAlignment(Pos.CENTER_RIGHT);
      root.add(btnPanel,1,3);
      GridPane.setMargin(btnPanel, new Insets(10, 0, 0, 0));

      Scene scene = new Scene(root);
      primaryStage.setScene(scene);
      primaryStage.show();
   }

}
```