Interfaces Java

Introduction à JavaFx

Introduction à JavaFX

Contenu :

- JavaFx, qu'est-ce que c'est?
- Hiérarchie des composants
- Le stage
- La scène
- Gestionnaires de disposition
- Noeuds

JavaFx, qu'est-ce que c'est ?

JavaFX est la nouvelle génération d'interfaces utilisateur graphique Java (GUI). Il s'agit d'une boîte à outils qui permet aux développeurs de construire rapidement des applications riches multi-plateformes. JavaFX permet de construire des interfaces combinant des graphiques, des animations et des contrôles d'interface utilisateur. La nouvelle JavaFX 8 est une interface de programmation d'application du langage Java pur (API). Le but de JavaFX est d'être utilisé dans de nombreux types de périphériques, tels que les systèmes embarqués, les smartphones, les téléviseurs, les ordinateurs tablettes et les ordinateurs de bureau.

Présentement, nous sommes à la version 8 de JavaFx, mais ne vous fiez pas à ce chiffre, il s'agit plutôt de la troisième mise-à-jour majeure du produit. Le 8 signifie que cette version est faite pour fonctionner de façon totalement intégrée avec la jdk 8 de Java. Sur le Web, vous trouverez de nombreux exemples, vous remarquez sans doute que la version 1 de JavaFx (la toute première) s'approche beaucoup plus du Html que du langage Java, c'est ce qui a fait (entre autres) de cette version un échec. La version 2, elle aussi très présente sur le Web, ressemble beaucoup à la version 8, mais il y a quand même des différences. Portez donc attention aux exemples que vous consultez.

Bref, pour fonctionner de façon complètement intégrée avec votre environnement de développement Eclipse ou NetBeans, il est préférable de travailler avec l'API 8 de Java (jdk) et la version 8 de JavaFx. Il n'y a aucun autre composant à installer sauf si vous voulez profiter des librairies pour les boîtes de dialogue ou utiliser le module GUI qui est la façon idéale de travailler avec JavaFx, mais que vous verrez dans un prochain cours.

Dans ce cours, vous allez voir les bases pour construire manuellement une interface complète pour interargir avec l'utilisateur.

Hiérarchie des composants

Une application JavaFX est composée de plusieurs éléments.

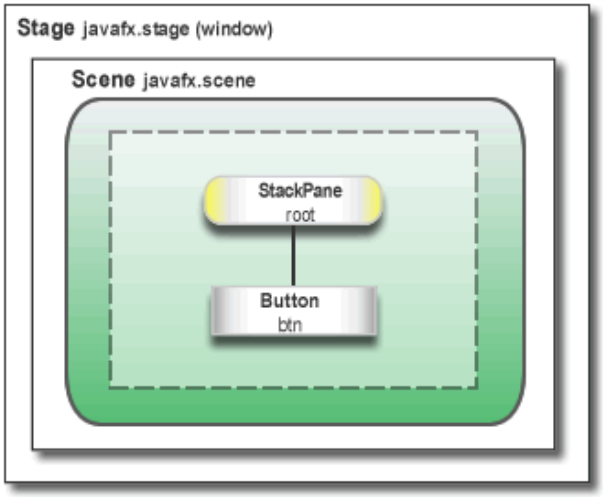
A. D'abord, la classe **Application** de laquelle hérite toute interface JavaFX, c'est donc le point d'entrée des applications JavaFx. Voici les opérations effectuées, une fois une instance de cette classe créée :
   1. Appel de la méthode init()
   2. Appel de la méthode start(javafx.stage.Stage)
   3. Attente de la fin de l'application à la rencontre de l'un de ces événements :
      - Un appel à la méthode Platform.exit()
      - La dernière fenêtre de l'application est fermée implicitExit l'attribut on Platform est vrai.
      - Un Appel de la méthode stop().
   4. Pour être lancée, dans le cadre de votre cours, vous aurez besoin de la méthode main ainsi :

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

B. Ensuite, le premier élément dont est constituée une **Application** FX est un objet **Stage**, (on peut s'imaginer une salle de spectacle), c'est dans cet objet que tout se passe, il représente la fenêtre de notre application.

C. À l'intérieur de cet objet Stage il y a un objet **Scene**, (la scène dans une salle). Tout ce qui apparait dans l'application y est inséré.

D. Enfin, l'objet Scene contient des éléments graphiques, (les acteurs). Ces éléments graphiques sont des objets qui peuvent être de différents types : des cercles, des rectangles, des composants, des images, des contrôles d'interaction (bouton, zone de texte, etc.).



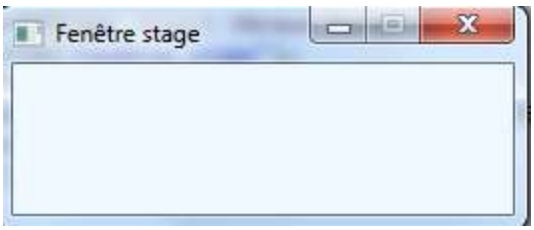
Le stage

La classe Stage, c'est notre fenêtre, celle qui reçoit la scène et tous ses composants. Le stage est reçu en paramètre par la méthode start de la classe Application.

```
public class HelloWorld extends Application
{
    @Override public void start(Stage stage)
    {
```

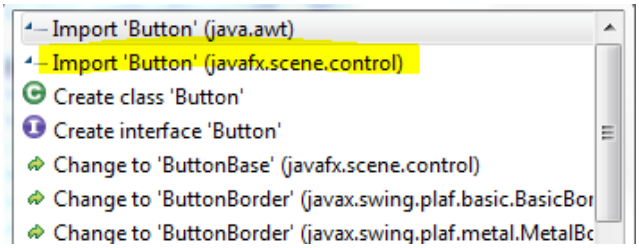
Les attributs que l'on peut ajuster sur le Stage sont

- sa taille, (sizeToScene, setMaxWidth, setMinWidth, setResizable…)
- le contenu de sa barre de titre, (setTitle)
- l'ajout de la scène, (setScene)
- l'affichage à l'écran (show)




```
public class ExempleStage extends Application
{
	private Scene scene;
	private BorderPane root;

	public void start(Stage stage)
	{
		root = new BorderPane();
		scene = new Scene(root, 250, 75, Color.ALICEBLUE);
		stage.setTitle("Fenêtre stage");
		stage.setScene(scene);
		stage.setResizable(false);
		stage.show();
	}
	public static void main(String[] args)
	{
		Application.launch(args);
	}
}
```

Observez la méthode main pour permettre à votre application de s'exécuter.



Mise en garde : quand vous ajoutez des éléments pour votre interface, vous devez régulièrement importer des librairies pour utiliser vos composants. Certains composants existent dans plusieurs librairies, il est **important de choisir le composant provenant de la librairie JavaFx** si vous voulez que votre application fonctionne. Dans notre exemple, il faut choisir le deuxième choix pour ajouter un bouton et non pas le premier.

La scène

Comme nous avons dit, la scène est l'endroit qui accueille les différents éléments d'interface que vous ajoutez. C'est un arbre hiérarchique de noeuds qui représente tous les éléments visuels de l'interface utilisateur.

Un élément dans une scène graphique est appelé un noeud. Chaque noeud a un ID et une classe de style. À l'exception du noeud racine de la scène, chaque noeud a un parent et peut avoir des enfants.

Mais, un nœud peut aussi avoir :

- Effets, comme taches et ombres
- Opacité
- Transformation
- Écouteurs d'événement

Une Scène doit contenir un nœud racine (root), en effet il y a un nœud de départ duquel est issu tous les autres. Le nœud racine est généralement un gestionnaire de disposition (exemple : un BorderPane) qui nous aide à disposer les descendants du nœud racine dans l'écran.

Les attributs que l'on peut ajuster sur la Scène sont :

- sa taille,
- la couleur de son arrière-plan,
- son composant racine,
- sa position
- les événements qui surviennent.

Voici les différents constructeurs d'une scène :

Constructor and Description
Scene(**Parent** root) Creates a Scene for a specific root Node.
Scene(**Parent** root, double width, double height) Creates a Scene for a specific root Node with a specific size.
Scene(**Parent** root, double width, double height, boolean depthBuffer) Constructs a scene consisting of a root, with a dimension of width and height, and specifies whether a depth buffer is created for this scene.
Scene(**Parent** root, double width, double height, **Paint** fill) Creates a Scene for a specific root Node with a specific size and fill.
Scene(**Parent** root, **Paint** fill) Creates a Scene for a specific root Node with a fill.

Composant racine

Le composant racine de la scène est important car c'est à partir de celui-ci que l'on peut ajouter tous les autres composants. Ordinairement, le composant racine est un gestionnaire de disposition que vous verrez dans la prochaine section et c'est celui-ci qui va déterminer comment les composants vont être ajoutés dans la scène. Selon le gestionnaire, on fera appel à la méthode **add** ou **addAll** ou une méthode set… Avant d'utiliser la méthode add ou addAll, il faut obtenir une référence sur le contenu du gestionnaire et cela se fait au moyen de l'instruction **getChildren()**. Observez l'exemple suivant d'une scène simple avec un élément racine AnchorPane.


```
public class ExempleScene extends Application
{
	private AnchorPane root;
	private Scene scene;
	private Circle circle;
	private Text text;
	private Font font;

	@Override
	public void start(Stage stage)
	{
		circle = new Circle(60, 40, 30, Color.GREEN);
		text = new Text(10, 90, "JavaFX Scene");
		text.setFill(Color.DARKRED);
		font = new Font(20);
		text.setFont(font);
		root = new AnchorPane();
		root.getChildren().add(circle);
		root.getChildren().add(text);
		scene = new Scene(root, 200, 150);
		scene.setFill(Color.LIGHTGRAY);

		stage.setScene(scene);
		stage.show();
	}

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

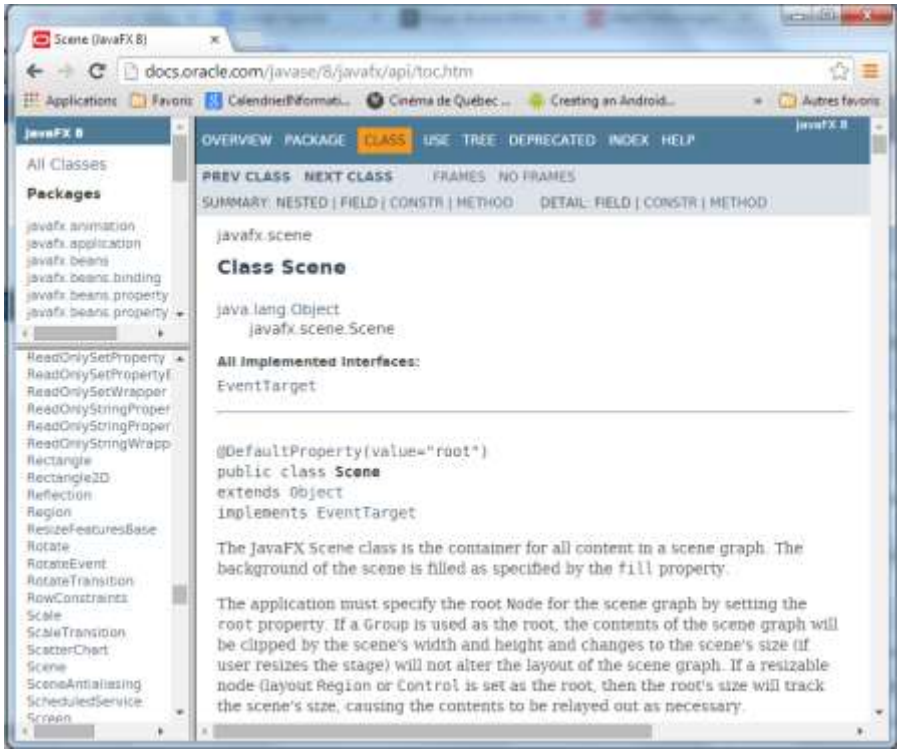
En bref :

- La classe *Application* est le point d'entrée des applications JavaFx. Elle reçoit un « stage » par sa méthode Start.
- Le premier élément dont est constituée une application FX est un objet **Stage**, (salle), c'est dans cet objet que tout se passe, il représente la fenêtre de notre application.
- À l'intérieur de cet objet **Stage** il y a un objet **Scene**, (la scène dans une salle). Tout ce qui apparait dans l'application y est inséré.

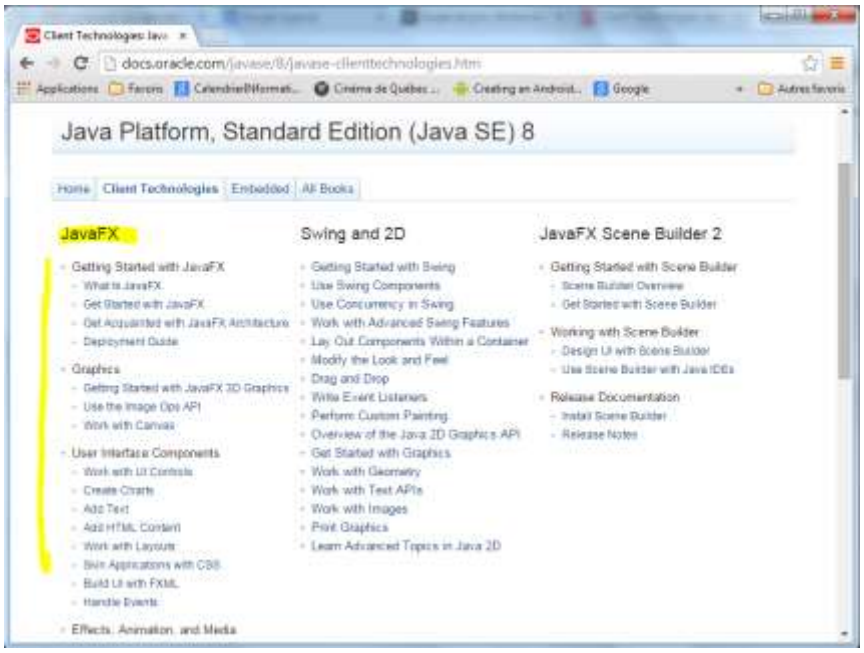
- Enfin, l'objet **Scene** contient des éléments graphiques, (les acteurs). Ces éléments graphiques sont des objets qui peuvent être de différents types, mais il y a un seul composant racine qui contient tous les autres.
- Pour exécuter une Application il faut une méthode main et faire appel à l'instruction launch pour démarrer. (`Application.launch(args);`)

Maintenant que nous savons comment créer une fenêtre grâce aux objets **Stage** et **Scene** et que nous savons comment afficher cette fenêtre grâce à l'objet Application, nous allons nous intéresser à comprendre comment organiser les composants dans une fenêtre à l'aide des gestionnaires de disposition.

Mais, avant toute chose, il est important de comprendre, qu'il est impossible ici de tout dire sur tous les composants. Par exemple, que doit-on faire pour modifier la couleur d'arrière-plan d'une scène ? Comment faire pour modifier la taille de la scène ? Il faut prendre l'habitude de consulter régulièrement **l'API** de JavaFX 8 que vous trouvez ici http://docs.oracle.com/javase/8/javafx/api/toc.htm. Vous trouverez là toute l'information nécessaire pour construire des interfaces JavaFX intéressantes.



De plus, le site d'Oracle offre un **tutoriel** très complet et bien fait où vous pourrez parfaire vos connaissances. Vous le trouverez à cette adresse :

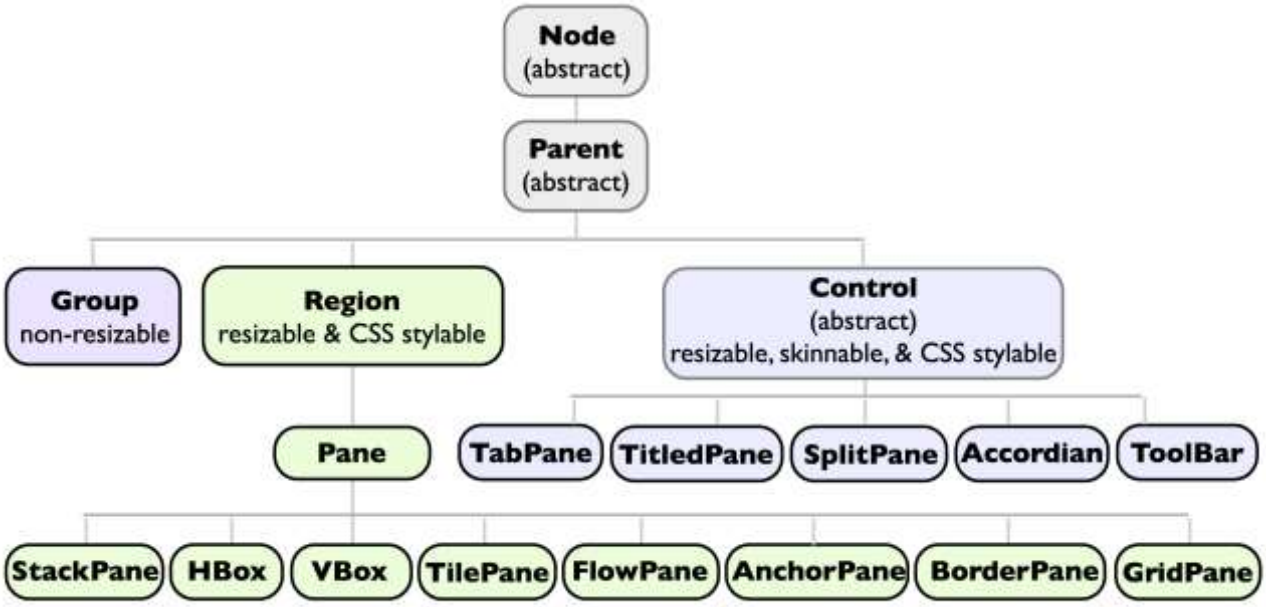


Gestionnaires de disposition

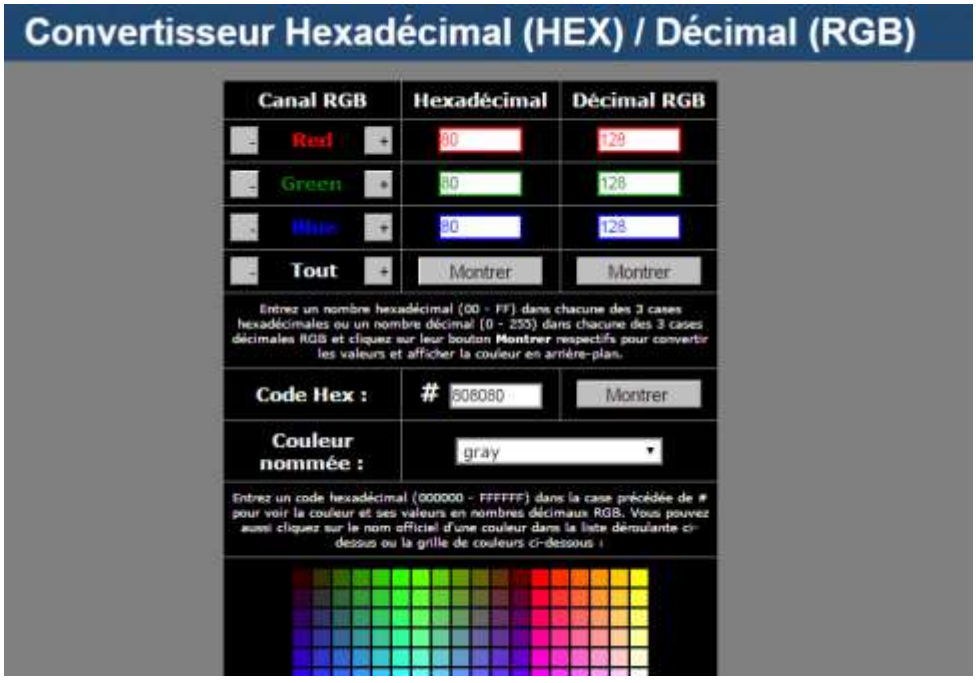
Afin de disposer facilement les composants, JavaFx dispose de gestionnaires de disposition, les « panes » tous issus de la classe *Region*.

Une application JavaFX peut disposer manuellement ses contrôles en mettant la position et les propriétés de taille pour chaque élément. Cependant, une option plus facile se sert de gestionnaires de disposition. JavaFX fournit plusieurs gestionnaires de disposition pour la configuration facile et la gestion de dispositions classiques comme des rangs, des colonnes, des grilles, etc. Quand la fenêtre est redimensionnée, le gestionnaire de disposition replace automatiquement et redimensionne les nœuds qu'il contient selon les propriétés de ceux-ci.

L'image ci-dessous présente les gestionnaires de dispositions qui sont tous ceux qui paraissent en vert. Vous remarquez qu'ils sont tous issus de la classe Region. On remarque aussi que les gestionnaires ont comme parent direct l'objet Pane. Un Pane peut accueillir du contenu, mais est plutôt utilisé pour des composants graphiques (cercle, rectangle, dessin, images, etc.). Nous allons étudier quelques-uns des gestionnaires les plus utilisés soit : BorderPane, AnchorPane, HBox, VBox et GridPane, mais sachez qu'il y en a d'autres qui existent et la documentation d'Oracle vous aidera à choisir celui qui convient le mieux à votre situation.

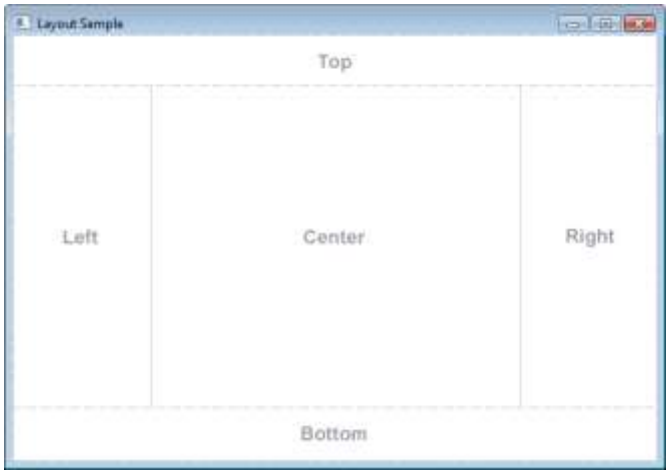


Tous les gestionnaires que nous allons voir offrent des fonctions de base identiques. On peut modifier la **distance** des composants entre eux grâce à la méthode `setSpacing`. On peut modifier la distance des quatre côtés du gestionnaire grâce à `setPadding` et on peut modifier l'apparence de l'arrière-plan en appliquant un style ainsi : `monHbox.setStyle("-fx-background-color: #333333;")`, pour obtenir une couleur de fond. Pour connaître le code hexadécimal d'une couleur, consultez la page suivante http://www.proftnj.com/RGB3.htm.

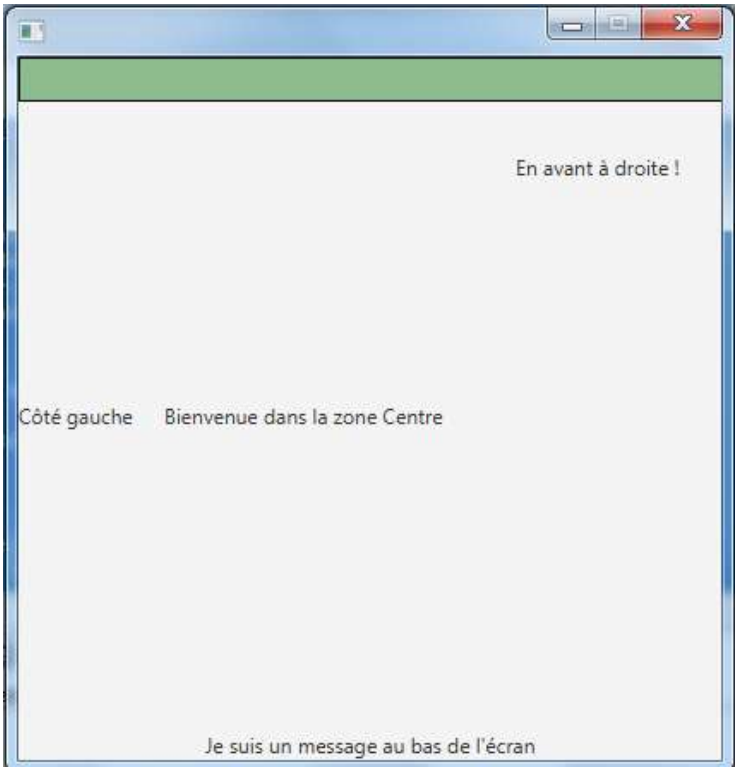


BorderPane

BorderPane est divisé en 5 régions et peut accueillir **un seul composant par région**. C'est la raison pour laquelle la façon d'ajouter un composant est différente, tous les autres gestionnaires ont une méthode commune d'ajout. Pour le BorderPane on utilise la méthode setCenter, setLeft, setRight, setBottom ou setTop puisqu'on ne peut ajouter qu'un seul composant par région. Si on ajoute un deuxième élément au même encroit dans un BorderPane, le deuxième écrase le premier.

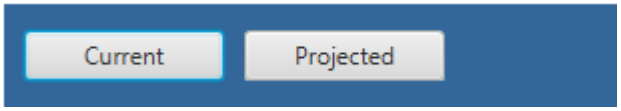


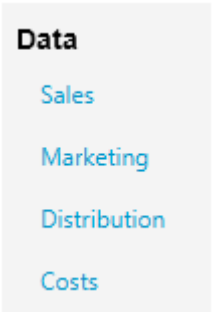
Exemple : ExempleBorderPane



```java
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class ExempleBorderPane extends Application
{
	// Attributs de la classe On pourra les réutiliser dans d'autres méthodes
	private BorderPane root;
	private Rectangle rectangle;
	private Label labelGauche, labelDroit, labelCentre, labelBas;

	@Override
	public void start(Stage stage)
	{
		// Haut
		rectangle = new Rectangle(410, 25, Color.DARKSEAGREEN);
		rectangle.setStroke(Color.BLACK);

		// Gauche
		labelGauche = new Label("Côté gauche");
		// alignement centre horizontal et vertical
		BorderPane.setAlignment(labelGauche, Pos.CENTER);

		// Droit
		labelDroit = new Label("En avant à droite !");
		// distance du composant H D B G
		labelDroit.setPadding(new Insets(30, 25, 0, 25));

		// Centre
		labelCentre = new Label("Bienvenue dans la zone Centre");

		// Bas
		labelBas = new Label("Je suis un message au bas de l'écran");
		BorderPane.setAlignment(labelBas, Pos.CENTER);

		root = new BorderPane();
		root.setTop(rectangle);
		root.setLeft(labelGauche);
		root.setRight(labelDroit);
		root.setCenter(labelCentre);
		root.setBottom(labelBas);
		stage.setResizable(false);
		stage.setScene(new Scene(root, 400, 400));
		stage.show();
	}
	public static void main(String[] args)
	{
		launch(args);
	}

}
```

HBox et VBox





Ces deux gestionnaires disposent horizontalement (sur une seule ligne) ou verticalement (sur une seule colonne) les composants les uns par rapport aux autres. L'ordre des composants est celui de l'ajout. On ajoute les éléments un à un ou plusieurs à la fois de ces façons :

```
monHbox.getChildren().add(monComposant);
monHbox.getChildren().addAll(monComposant1, monComposant2…);
```

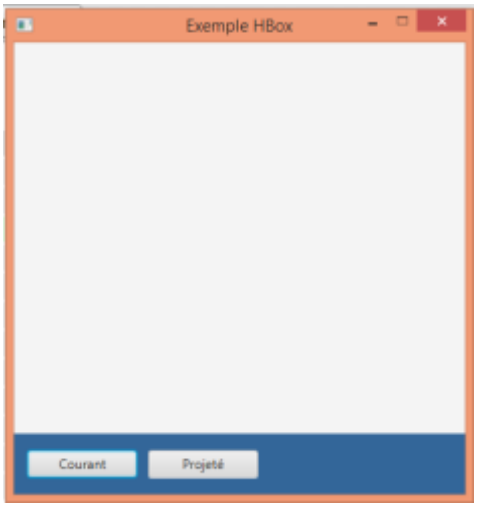
Remarquez l'instruction getChildren qui permet d'obtenir une référence sur les éléments de la zone de contenu du gestionnaire de disposition.

Exemple : ExempleVBox


```
public class ExempleVBox extends Application
{
	private BorderPane root;
	private VBox vbox;
	private Text titre;
	private ListView<String> liste;
	private ObservableList<String> items;
	private Label label;

	@Override
	public void start(Stage stage)
	{
	    titre = new Text("Données");
	    titre.setFont(Font.font("Arial", FontWeight.BOLD, 14));

	    //zone de liste qui sera expliquée ultérieurement
	    liste = new ListView<String>();
	    items = FXCollections.observableArrayList (
	        "Un", "Deux", "Trois", "Quatre");
	    liste.setItems(items);
	    liste.setMaxHeight(100);
	    label = new Label("Bonjour");

	    vbox = new VBox();
	    vbox.setPadding(new Insets(10));
	    vbox.setSpacing(8);
	    //ajout dans le gestionnaire vBox
	    vbox.getChildren().addAll(titre, liste, label);

	    //ajout du vbox dans le composant racine
	    root = new BorderPane();
	    root.setCenter(vbox);
	    stage.setResizable(false);
```

```
            stage.setScene(new Scene(root, 400, 400));
            stage.setTitle("Exemple VBox");
            stage.show();
        }

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

Exemple : ExempleHBox




```
public class ExempleHBox extends Application
{

        private BorderPane root;
        private HBox hbox;
        private Button bouton, bouton2;

        @Override
        public void start(Stage stage)
        {
            bouton = new Button("Courant");
            bouton.setPrefSize(100, 20);

            bouton2 = new Button("Projeté");
            bouton2.setPrefSize(100, 20);

            hbox = new HBox();
            hbox.setPadding(new Insets(15, 12, 15, 12));
            hbox.setSpacing(10);
            hbox.setStyle("-fx-background-color: #336699;");
            hbox.getChildren().addAll(bouton, bouton2);

            // ajout du hbox dans le composant racine
            root = new BorderPane();
            root.setBottom(hbox);
            stage.setResizable(false);
            stage.setScene(new Scene(root, 400, 400));
            stage.setTitle("Exemple HBox");
            stage.show();
        }

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

Maintenant que vous connaissez 3 gestionnaires de disposition, vous pouvez imaginer facilement les combiner. En voici un exemple.

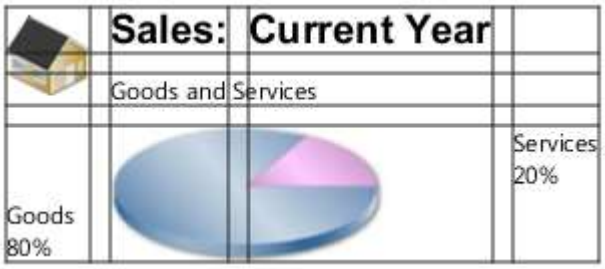
Exemple : ExempleBorderEtBox


```
public class ExempleBorderEtBox extends
Application
{
	private BorderPane root;
	private HBox hbox;
	private VBox vbox;
	private Button bouton, bouton2;
	private Text titre;
	private ListView<String> liste;
	private ObservableList<String> items;
	private Label label;

	@Override
	public void start(Stage stage)
	{
		creerZoneBas();
		creerZoneCentre();
		root = new BorderPane();
		root.setBottom(hbox);
		root.setCenter(vbox);
		stage.setResizable(false);
		stage.setScene(new Scene(root, 400, 400));
		stage.setTitle("Exemple Combinés");
		stage.show();
	}

	private void creerZoneBas()
	{
		bouton = new Button("Courant");
		bouton.setPrefSize(100, 20);
		bouton2 = new Button("Projeté");
		bouton2.setPrefSize(100, 20);

		hbox = new HBox();
		hbox.setPadding(new Insets(15, 12, 15, 12));
		hbox.setSpacing(10);
		hbox.setStyle("-fx-background-color: #336699;");
		hbox.getChildren().addAll(bouton, bouton2);
	}

	private void creerZoneCentre()
	{
	    titre = new Text("Données");
	    titre.setFont(Font.font("Arial", FontWeight.BOLD, 14));

	    //zone de liste qui sera expliquée ultérieurement
	    liste = new ListView<String>();
	    items =FXCollections.observableArrayList (
```

```
                "Un", "Deux", "Trois", "Quatre");
            liste.setItems(items);
            liste.setMaxHeight(100);
            label = new Label("Bonjour");

            //ajout dans le gestionnaire vBox
            vbox = new VBox();
            vbox.setPadding(new Insets(10));
            vbox.setSpacing(8);
            vbox.getChildren().addAll(titre, liste, label);
        }

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

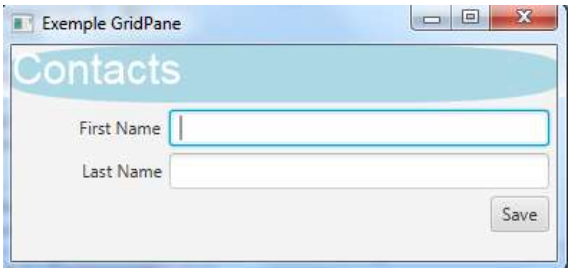
GridPane

GridPane permet de créer une grille flexible de lignes et de colonnes dans laquelle on place des nœuds. Les nœuds peuvent être placés dans n'importe quelle cellule de la grille et peuvent s'étendre sur plusieurs cellules en fonction des besoins. Un **GridPane** est utile pour créer des formulaires ou n'importe quel agencement qui est organisé en lignes et en colonnes.



Exemple : ExempleGridPane

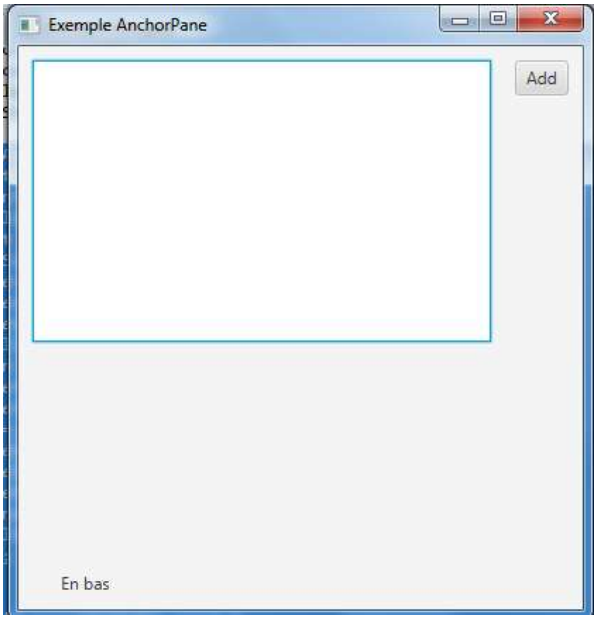
```
public class ExempleGridPane extends Application
{
        private BorderPane root;
        private Scene scene;
        private GridPane gridpane;
        private ColumnConstraints column1, column2 ;
        private Label fNameLbl, lNameLbl;
        private TextField fNameFld, lNameFld;
        private Button saveButton;
        private HBox topBanner;
        private Text contactText;

        public void start(Stage stage)
        {
                //Crée les composants atomiques
                contactText = new Text("Contacts");
                contactText.setFill(Color.WHITE);
                Font serif = Font.font("Dialog", 30);
                contactText.setFont(serif);
                fNameLbl = new Label("First Name");
                fNameFld = new TextField();
```



```
            lNameLbl = new Label("Last Name");
            lNameFld = new TextField();
            saveButton = new Button("Save");

            //Crée la partie du haut
            topBanner = new HBox();
            topBanner.setPrefHeight(40);
            //style pour obtenir la zone bleu
            String backgroundStyle = "-fx-background-color: lightblue;"
                        + "-fx-background-radius: 30%;" ;
            topBanner.setStyle(backgroundStyle);
            topBanner.getChildren().add(contactText);

            // Crée une colonne de 100 pixels
            column1 = new ColumnConstraints(100);
            //Crée une colonne de largeur min 50, largeur préférée 150 et largeur max 300
            column2 = new ColumnConstraints(50, 150, 300);
            column2.setHgrow(Priority.ALWAYS);

            gridpane = new GridPane();
            gridpane.getColumnConstraints().addAll(column1, column2);
            gridpane.setPadding(new Insets(5));
            gridpane.setHgap(5);
            gridpane.setVgap(5);
            //Ajouter les composants dans la grille à col, ligne
            gridpane.add(fNameLbl, 0, 0);
            gridpane.add(lNameLbl, 0, 1);
            gridpane.add(fNameFld, 1, 0);
            gridpane.add(lNameFld, 1, 1);
            gridpane.add(saveButton, 1, 2);

            // Ajuster l'alignement
            GridPane.setHalignment(fNameLbl, HPos.RIGHT);
            GridPane.setHalignment(lNameLbl, HPos.RIGHT);
            GridPane.setHalignment(fNameFld, HPos.LEFT);
            GridPane.setHalignment(lNameFld, HPos.LEFT);
            GridPane.setHalignment(saveButton, HPos.RIGHT);

            root = new BorderPane();
            root.setTop(topBanner);
            root.setCenter(gridpane);
            scene = new Scene(root, 380, 150, Color.WHITE);
            stage.setScene(scene);
            stage.setTitle("Exemple GridPane ");
            stage.show();
      }

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

AnchorPane

AnchorPane est le dernier gestionnaire de disposition que nous allons étudier. C'est le gestionnaire par défaut utilisé lorsqu'on travaille avec le module GUI. Ce gestionnaire permet d'ancrer les nœuds vers le haut, le bas, la gauche, la droite, le centre. Quand la fenêtre est redimensionnée, les nœuds maintiennent leur position par rapport à leur point d'ancrage. Les nœuds peuvent être ancrés à plus d'une position et plus **d'un nœud peuvent être ancrés à la même position**. C'est sa principale différence avec le BorderPane.



Exemple : ExempleAnchorPane

```
public class ExempleAnchorPane extends Application
{
	private AnchorPane anchorPane;
	private ListView list;
	private Button button;
	private Label label;

	public void start(Stage stage)
	{
		//Crée les composants atomiques
		list = new ListView();
		list.setPrefSize(20, 200);
		button = new Button("Add");
		label = new Label("En bas");

		// List should stretch as anchorPane is
resized
		AnchorPane.setTopAnchor(list, 10.0);
		AnchorPane.setLeftAnchor(list, 10.0);
		AnchorPane.setRightAnchor(list, 65.0);

		// Button will float on right edge
		AnchorPane.setTopAnchor(button, 10.0);
		AnchorPane.setRightAnchor(button, 10.0);

		AnchorPane.setBottomAnchor(label, 10.0);
		AnchorPane.setLeftAnchor(label, 30.0);
		anchorPane = new AnchorPane();
		anchorPane.getChildren().addAll(list, button, label);
		stage.setScene(new Scene(anchorPane, 400, 400));
		stage.setTitle("Exemple AnchorPane");
		stage.show();
	}

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

}
```