Tutoriel sur les bases avec JavaFX

Par Mik Arber

Date de publication : 1 avril 2015

TOUT PUBLIC

Durée : 10 minutes

JavaFX est la bibliothèque graphique remplaçante de Swing et de AWT. Elle a pour avantage d'être utilisable via un langage objet et statiquement typé.

Pour réagir au contenu de cet article, un espace de dialogue vous est proposé sur le forum **Commentez**.

I - Introduction

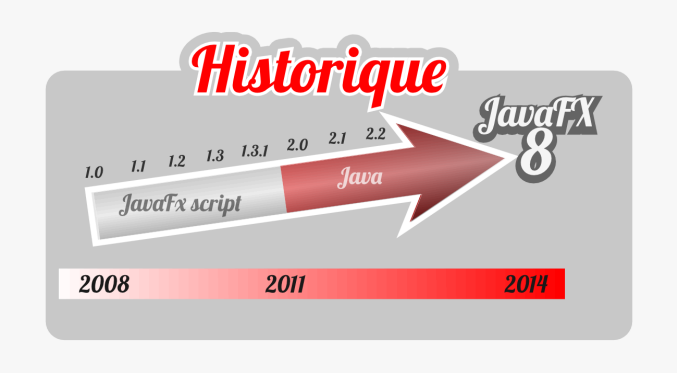
JavaFX est une bibliothèque graphique intégrée dans le **JRE** et le **JDK** de **Java**. **Oracle** la décrit comme « The Rich Client Platform », c'est-à-dire qu'elle permet de réaliser des interfaces graphiques évoluées et modernes grâce à de nombreuses fonctionnalités, telles que les animations, les effets, la 3D, l'audio, la vidéo, etc. Elle a de plus l'avantage d'être dans le langage Java, qui permet de réaliser des architectures avec des paradigmes objet, et aussi de pouvoir utiliser le typage statique.

Dans ce premier tutoriel, nous allons voir ensemble un rapide historique de la bibliothèque pour ensuite découvrir les fondamentaux que sont les classes « **Stage** », « **Scene** », « **Application** » et le « threading » associé, pour finir nous verrons les « **Node** » avec un exemple d'utilisation du « scene graphe ». Cette présentation ne fait pas dans le bling-bling, même si **JavaFX** est doué pour cela, en préférant se focaliser sur les concepts primordiaux d'une telle bibliothèque. Bien comprendre ces basiques vous aidera à bien commencer pour ensuite pouvoir faire des interfaces de qualité et peut-être spectaculaires.

II - Historique

La première version de **JavaFX** 1.0 a été créée en 2008. Elle était bien différente de la version courante 8, car elle était utilisable via un langage script spécifique. Il fallait aussi la télécharger à part et le support des outils était faible. A contrario maintenant la maturité aidant, elle est incluse par défaut dans **Java** et directement accessible dans les **IDE**. De plus, il est inutile d'apprendre un nouveau langage, car l'**API** est comme **Swing** et **AWT** en **Java**.

Cette information peut s'avérer pertinente, car vous pouvez encore trouver sur Internet des documents ou autres exemples de ces anciennes versions obsolètes qu'il ne faut pas utiliser. Le langage **Java** a été introduit à partir de la version 2.0.



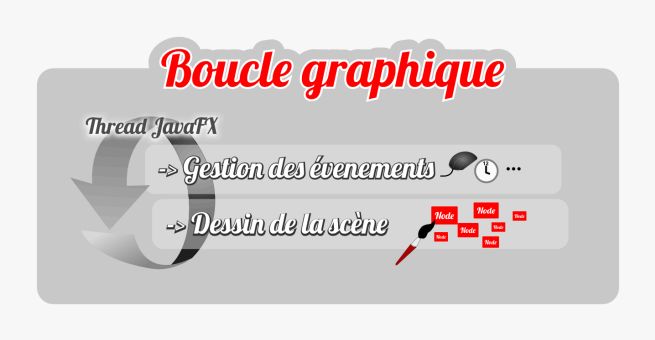
Une autre information importante est que **JavaFX** est la bibliothèque remplaçante de **Swing** qui elle-même remplaçait **AWT**. Dans les versions courantes 1.8, les trois bibliothèques sont toujours accessibles, mais seule **JavaFX** va, dans le futur, avoir de nouvelles fonctionnalités.

Les deux autres sont considérées comme des branches mortes. Donc si vous partez sur de nouveaux développements d'IHM en Java, je vous conseille d'utiliser **JavaFX** qui est la seule à avoir le support d'**Oracle** pour les prochaines années.

III - Premières lignes de code

Commençons par le début, c'est-à-dire comment lancer une application de type **JavaFX**. Comme classiquement en Java, le démarrage est réalisé via une classe contenant une méthode statique portant le nom « **main** ». Par contre et contrairement à **Swing** ou à **AWT**, **JavaFX** vous oblige à étendre la classe « **javafx.application.Application** ». Cela a pour but de forcer les développeurs à utiliser **JavaFX** dans le **Thread JavaFX**. En effet comme la grande majorité des bibliothèques graphiques, l'utilisation de **JavaFX** est monothread, c'est-à-dire que tous les appels à l'API

ou la création d'objets de type **JavaFX** doivent être réalisés dans ce thread. Le schéma, ci-dessous, vous indique de manière simplifiée le travail du thread.



Attention si vous utilisez l'**API** en dehors de ce thread votre IHM risque fortement d'avoir un comportement incongru ceci agrémenté de levées d'exceptions qui peuvent être difficiles à comprendre.

Ci-dessous, vous trouverez un exemple qui n'affiche rien, mais qui permet de comprendre la notion d'application ainsi que de comprendre le threading. Maintenant que **JavaFX** est inclus dans le **JDK**, il n'y a plus de contrainte particulière pour l'utiliser. Utilisez votre **IDE** préféré comme **Eclipse**, **Netbeans** ou **IntelliJ**. Configurez bien l'utilisation d'un JDK supérieur à la version 1.8 et c'est tout.

```
import javafx.application.Application;
import javafx.stage.Stage;


public class MyFirstJfxApplication extends Application{

    public static void main(String[] args) {
        System.out.println( "Main method inside Thread : " +  Thread.currentThread().getName());
        launch(args);
    }

    @Override
    public void start(Stage args) throws Exception {
        System.out.println( "Start method inside Thread : " +  Thread.currentThread().getName());
    }
}
```

Exécutez cette classe et vous devez avoir, dans la console, le texte suivant :

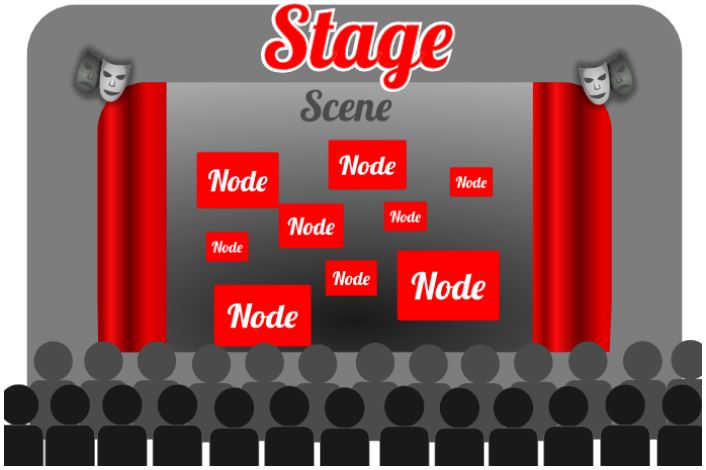
Main method inside Thread : main

Start method inside Thread : JavaFX Application Thread

IV - Bienvenue au théâtre

Écrire une bonne API est un art difficile. De plus c'est une activité réalisée par l'Homme pour l'Homme. Sachant cela, une bonne méthode de création est d'avoir recours à des analogies connues par le plus grand nombre d'entre nous. Pour leur part, les créateurs de **JavaFX** ont choisi des terminologies provenant du théâtre.

- **javafx.stage.Stage** : peut-être traduit par l'estrade. C'est le stage qui définit la taille de la fenêtre et aussi son titre. En **AWT** ou **Swing** cela correspondait à une **Frame** ou **Jframe**. C'est le lien avec le système de fenêtre des systèmes d'exploitation.
- **javafx.scene.Scene** : correspond au décor où a lieu l'action. Comme au théâtre il est possible d'avoir plusieurs décors pour une même estrade.
- **javafx.scene.Node** : ils n'ont pas poussé l'analogie, jusqu'au bout, en utilisant le terme acteur. Mais les nodes se placent dans le décor comme des acteurs.



V - Afficher vos premières nodes

Pour afficher vos premiers objets graphiques, il suffit de créer un « Scene Graph ». Derrière ce mot se cache juste un arbre d'éléments graphiques qui permet de définir, dans un repère, où vos éléments vont se dessiner et aussi dans quel ordre. Comme tout arbre, nous retrouvons les notions classiques d'élément de type parent ou de type feuille.

Pour les utilisateurs de **AWT** ou **Swing** cela existait déjà, avec des « **Component** », « **JComponent** » et des « **Container** ». La différence vient que cette fois **JavaFX** propose des primitives graphiques telles que le rectangle, le cercle, etc., et pas uniquement des gadgets graphiques comme le bouton, les listes de choix, etc.

Il y a plusieurs types de nodes :

javafx.scene.Canvas : nous n'allons pas voir ce type de nodes, mais sachez qu'il permet d'avoir un « graphic context » comme en Java2D, cela permet de réaliser des dessins très élaborés avec une facilité pour l'optimisation. De là, vous pouvez implémenter votre propre « Scene Graph » ;

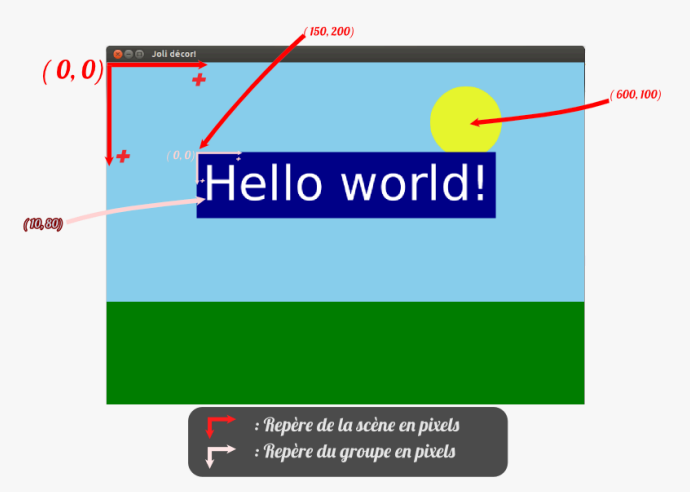
javafx.scene.ImageView : ce type de nodes sert à la manipulation et l'affichage d'images ;

javafx.scene.MediaView : celui-ci sert à afficher de la vidéo ou jouer des sons ;

javafx.scene.Parent : nous utiliserons un parent de type « Group » dans l'exemple suivant. Les parents servent à composer les éléments graphiques ;

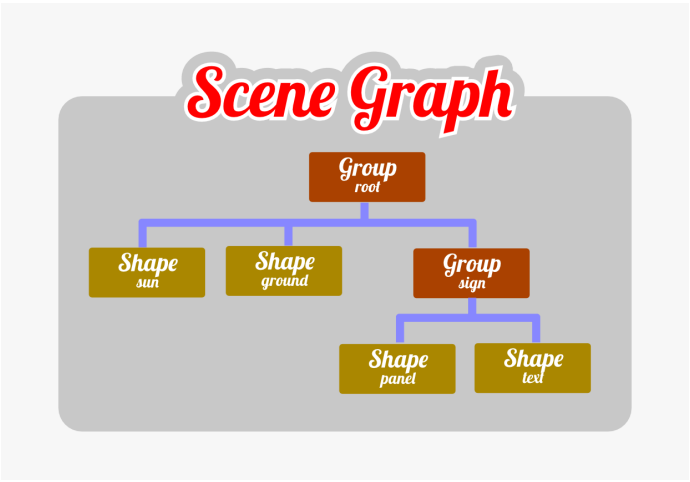
javafx.scene.Shape : pour illustrer le « Scene Graph », nous allons utiliser ce type de nodes, car ce sont les plus simples. Elles permettent d'afficher des éléments graphiques basiques tels que les rectangles, cercles, textes, etc.

Dans l'exemple de code, fourni plus bas, nous allons afficher l'application suivante composée d'une scène avec un fond bleu, contenant un soleil, un sol vert, un panneau lui-même composé de deux éléments : un panneau bleu foncé et un texte blanc.



Le système de coordonnées est de type « Device », c'est-à-dire que l'unité est le pixel, que l'origine est située en haut à gauche et que les sens positifs sont dirigés vers la **droite** et vers le **bas**. Ceci est très classique dans le monde du graphisme.

Pour réaliser cette application, une arborescence de nodes doit être créée correspondant au diagramme ci-dessous. Vous pouvez observer que pour la composition, il faut utiliser un parent de type Group. Ces enfants sont alors placés dans le repère du parent et non dans celui de la scène. Vous remarquerez aussi que l'ordre des nodes est utilisé pour définir l'ordre de dessin. C'est grâce à cela que le soleil est dessiné derrière le panneau.



Le code source est le suivant. Reprenez l'application précédente et remplacez la méthode « start » avec celle-ci :

```
    @Override
    public void start(Stage stage) throws Exception {
        // définit la largeur et la hauteur de la fenêtre
        // en pixels, le (0, 0) se situe en haut à gauche de la fenêtre
          stage.setWidth(800);
        stage.setHeight(600);
        // met un titre dans la fenêtre
        stage.setTitle("Joli décor!");

        // la racine du sceneGraph est le root
        Group root = new Group();
        Scene scene = new Scene(root);
        scene.setFill(Color.SKYBLUE);

        // création du soleil
        Circle sun = new Circle(60, Color.web("yellow", 0.8));
        sun.setCenterX(600);
```

```
        sun.setCenterY(100);

        // création du sol
        Rectangle ground = new Rectangle(0, 400, 800, 200);
        ground.setFill(Color.GREEN);

        // création d'un élément plus complexe, le panneau
        Group sign = new Group();
        sign.setTranslateX(150);
        sign.setTranslateY(200);
        // Attention les coordonnées sont celles du panneau, pas de la scène
        Text text = new Text(10, 30, "Hello world!");
        text.setFont(new Font(80));
        text.setFill(Color.WHITE);
        // le repère utilisé est celui du panneau
        Rectangle panel = new Rectangle( 0, -50, 500, 110);
        panel.setFill(Color.DARKBLUE);
        // composer l'élément plus complexe
        sign.getChildren().add(panel);
        sign.getChildren().add(text);

        // ajout de tous les éléments de la scène
        root.getChildren().add(sun);
        root.getChildren().add(ground);
        root.getChildren().add(sign);

        // ajout de la scène sur l'estrade
        stage.setScene(scene);
        // ouvrir le rideau
        stage.show();
    }
```

VI - Conclusion

Nous avons vu ensemble les concepts de **JavaFX** et l'analogie avec le théâtre. De plus nous avons regardé l'application **JavaFX** avec son threading ainsi que la composition du « Scene Graphe » par la création d'une arborescence de « Node ».

Même si ce que nous avons vu ensemble n'est pas spectaculaire, cela n'est pas moins important, car cela correspond aux basiques de **JavaFX**. Ces bases sont classiques et communes à de nombreuses bibliothèques graphiques. Il est donc important de bien les comprendre pour les professionnels du graphisme et autres amateurs éclairés.

Sur ces bases, il y a encore une grande richesse fonctionnelle à découvrir avec les effets, les animations, la 3D, les CSS, etc.

Pour aller plus loin :

http://docs.oracle.com/javafx/

https://java.developpez.com/faq/javafx/

http://fxexperience.com/

Les sites de l'auteur :

- **Open source** : **http://capcaval.org**
- **Blog** : **http://miksblog.capcaval.org**
- **BD de codeur** : **http://100pcpc.capcaval.org**

VII - Remerciements

Nous tenons à remercier **Bouye** pour sa relecture attentive de cet article et **Mickaël Baron** pour la mise au gabarit ainsi que **ClaudeLELOUP** pour sa correction orthographique.