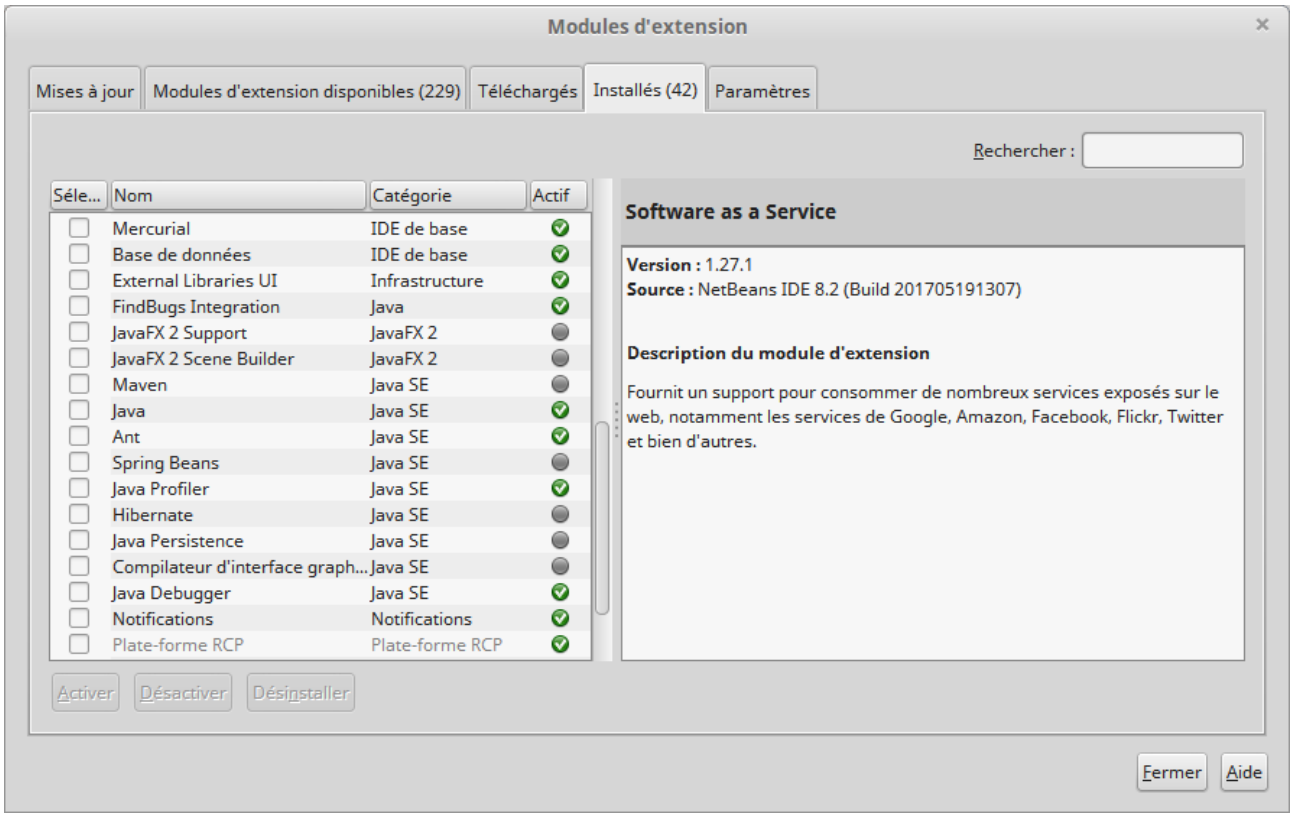
ING 1 - POO Java

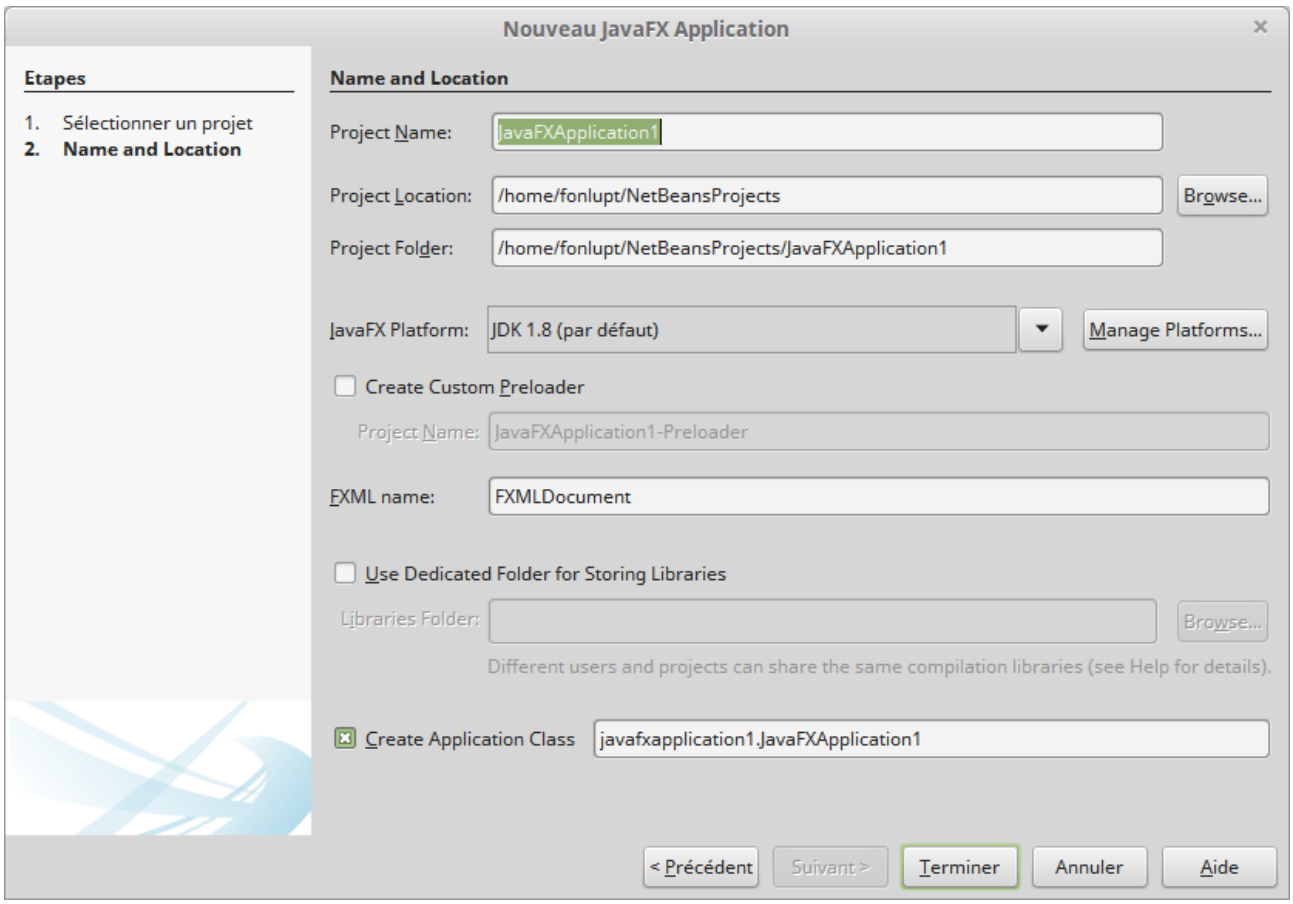
Autoformation à JavaFX

Installer JavaFX dans NetBeans

La première étape consiste à activer le support de JavaFX dans NetBeans. Pour ce faire, allez dans le menu Outils → Modules d'extension. Allez dans l'onglets des modules installés et activer javaFX2 Support et JavaFX2 Scene Builder. Il est possible que cela soit déjà installé dans votre version de NetBeans ou pas du tout téléchargé et dans ce cas qu'il soit nécessaire de le télécharger dans l'onglet Modules d'extensions disponibles.



Créer maintenant un nouveau projet JavaFX



Note : JavaFX est présent maintenant dans toutes les distributions Java depuis la version 8

Création d'une fenêtre

Pour nous familiariser avec l'environnement JavaFX, nous allons débuter par créer une fenêtre et lui affecter une couleur de fond.

Voici le code JavaFX permettant de créer cette fenêtre et de lui affecter une couleur de fond

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavafxApp1 extends Application {
   @Override
   public void start(Stage primaryStage) throws Exception {
      //création d'un groupe
      Group groupe = new Group();

      //Création de la Scene, on lui passe en argument la largeur et la hauteur
      Scene scene = new Scene(groupe ,600, 300);
```

```
        //couleur de fond pour la scene
        scene.setFill(Color.AZURE);

        //Titre su Stage
        primaryStage.setTitle("Application 1");

        //On passe la scene au Stage
        primaryStage.setScene(scene);

        //Affichage du stage
        primaryStage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

Le code a été tapé sous Netbeans

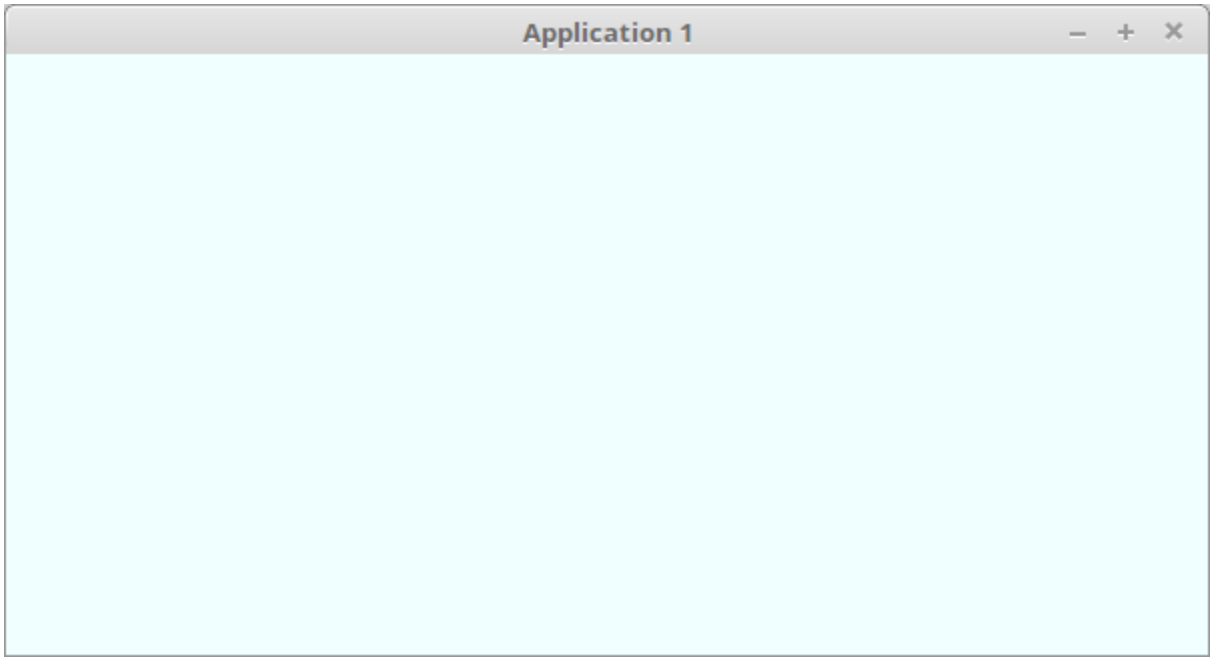
Quelques explications

La classe Group est un container de nœuds de JavaFX, elle permet de stocker facilement plusieurs nœuds de JavaFX. Group est une sous-classe de la classe Parent qui est elle-même une sous-classe de la classe Node (éléments de base de JavaFX)

Nous passons ensuite notre objet de type Group au constructeur de la classe Scene. Nous lui passons également la largeur et la hauteur de la fenêtre. On doit passer à l'objet de type Scene le nœud racine ou un objet de type Group ce que nous avons fait dans cet exemple.

La méthode setFill() permet de donner une couleur de fond. Nous donnons ensuite un nom au Stage et nous passons en argument au Stage un objet de type Scene. La dernière étape est d'appeler la méthode show() sur l'objet de type Stage.

Vous devriez obtenir la fenêtre suivante :



En utilisant la documentation de la classe Color de JavaFX, prenez une nouvelle couleur puis rendez la plus sombre

Dessin d'une ligne

Nous allons maintenant dessiner une ligne qui traverse l'écran en diagonale de haut à gauche à en bas à droite ; pour réaliser cette opération, vous utiliserez la classe Line en vous créerez un objet de cette classe.

Pour ajouter un ou plusieurs objets à notre objet de type Group, vous pouvez procéder de la manière suivante :

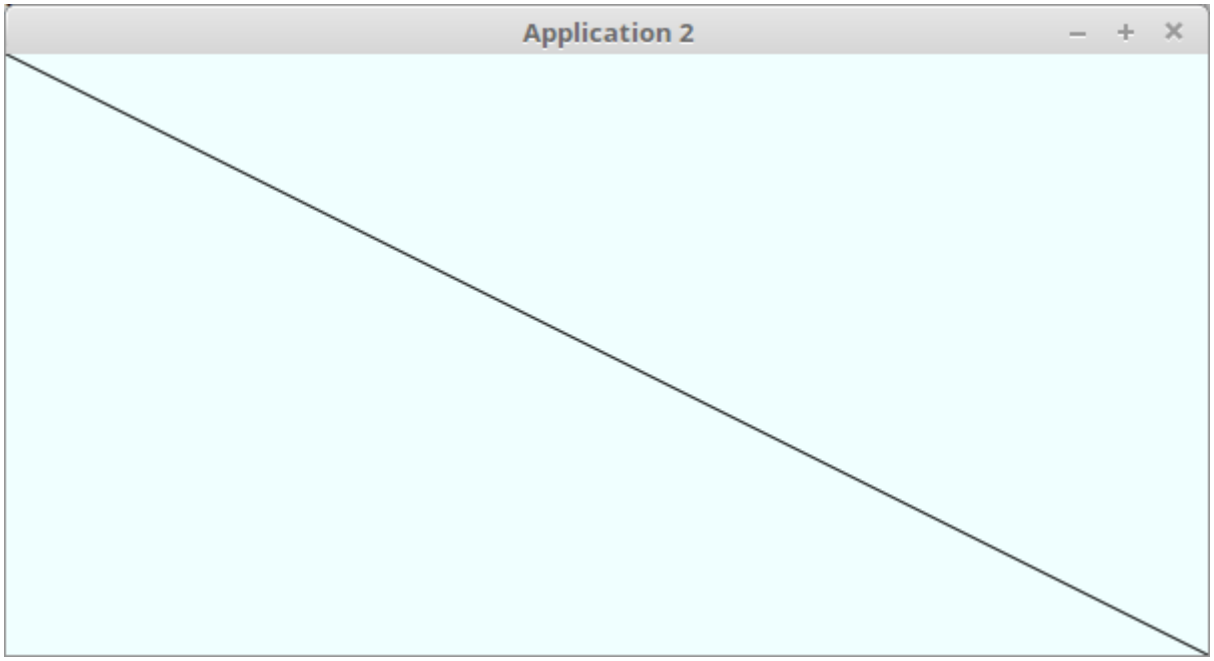
```
ObservableList<Node> liste = groupe.getChildren();

      liste.add(ligne);
```

Nous récupérons un objet de type ObservableList qui contient tous les nœuds enfants de l'objet groupe. Pour rajouter un élément, il suffit de l'ajouter avec la méthode add (c'est en fait une ArrayList).

Il faudra bien sûr rajouter les importations de Node et ObservableList si vous ne les avez pas mises

```
import javafx.scene.Node;
import javafx.collections.ObservableList;
```

Note : ligne est l'objet de type Line que nous venons de créer



Vous devriez obtenir quelque chose qui ressemble à la fenêtre ci-dessus.

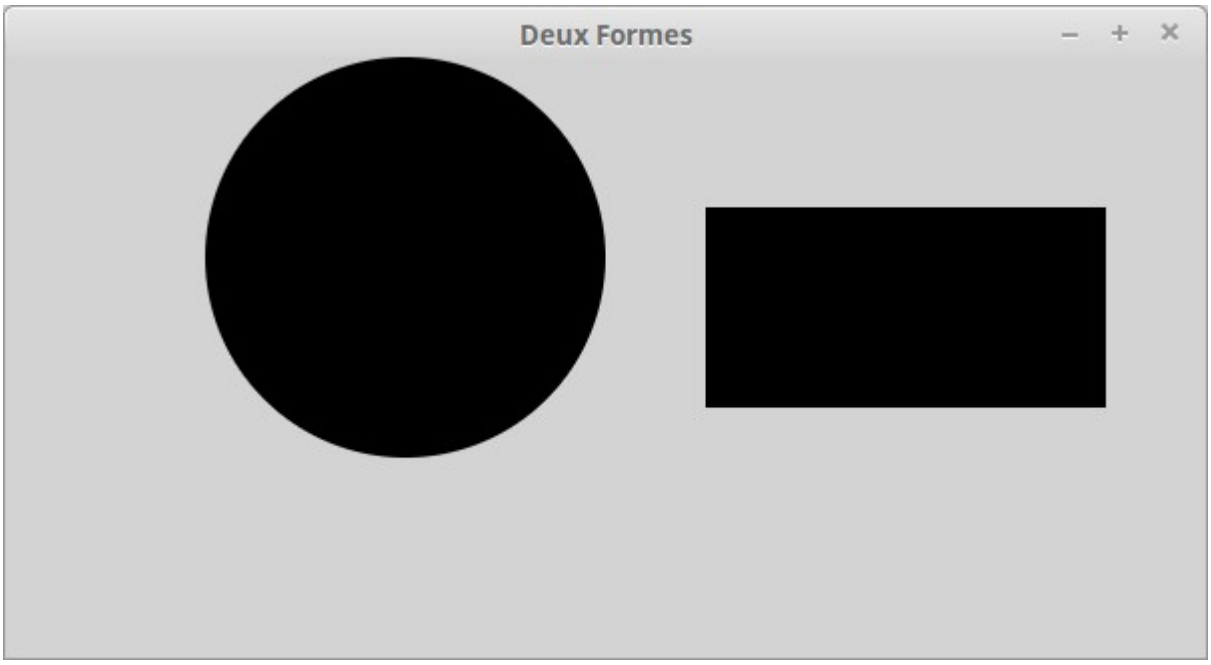
Une autre possibilité encore plus simple est d'utiliser la syntaxe suivante si le nœud root est défini comme étant un groupe :

```
root.getChildren().add(noeud);
```

Par exemple, les deux formes suivantes sont créées par le code suivant :

```
public class Formes extends Application {

    @Override
    public void start(Stage primaryStage){


        primaryStage.setTitle("Deux Formes");

        Group groupe = new Group();
        Scene scene = new Scene(groupe,600,300, Color.LIGHTGRAY);

        Rectangle rect1 = new Rectangle(350,75,200,100);

        Circle cerc1 = new Circle(200,100,100);

        groupe.getChildren().add(rect1);
        groupe.getChildren().add(cerc1);

        primaryStage.setScene(scene);
        primaryStage.show();




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


}
```

Vous obtenez alors la fenêtre suivante :



Comme vous pouvez le voir, les formes sont par défaut remplies de couleur noire. Nous allons maintenant ajouter un petit peu de couleur.

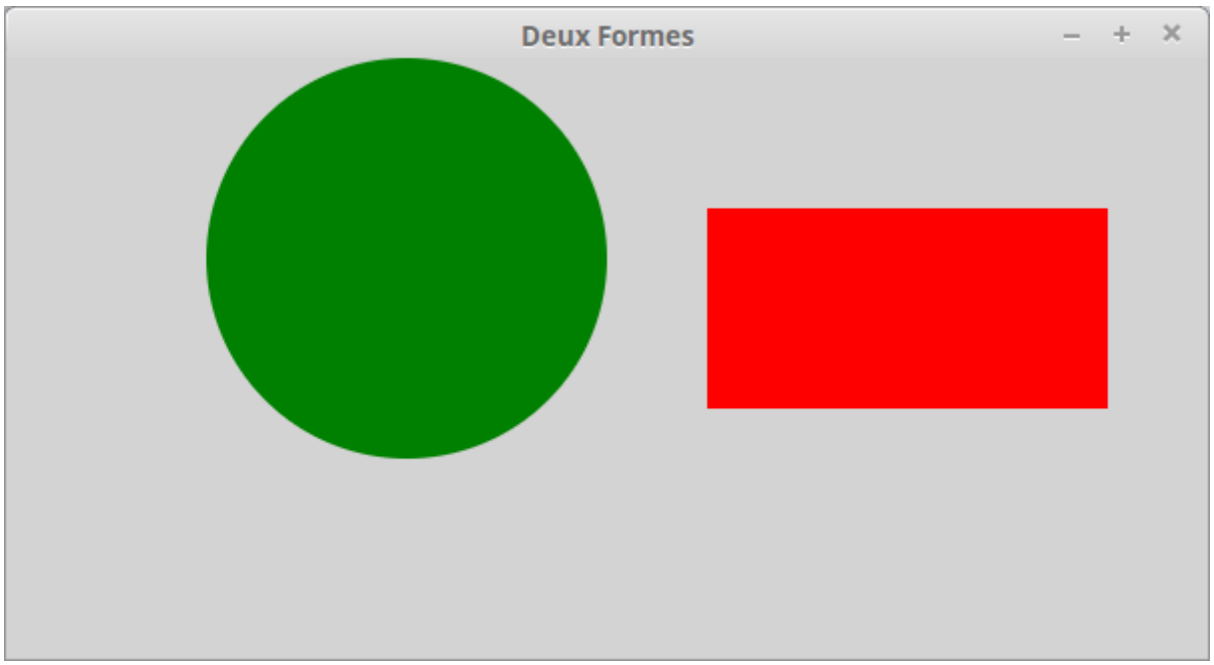
Si vous regardez attentivement, la classe Cercle par exemple, vous pouvez remarquer que celle-ci hérite de la classe Shape et que dans cette dernière sont définies des méthodes comme setFill (remplissage) et setStroke (contour). D'autres méthodes de la classe Shape permettent de définir l'épaisseur du contour, le type, etc.

Ces méthodes prennent un objet en paramètre de la classe Paint qui est une classe **abstraite** (vous vous souvenez du cours ?) qu'il faut donc absolument instancier dans une classe concrète. Pour le moment, nous nous limiterons à la classe Color mais on peut faire des choses plus complexes comme nous le verrons par la suite.

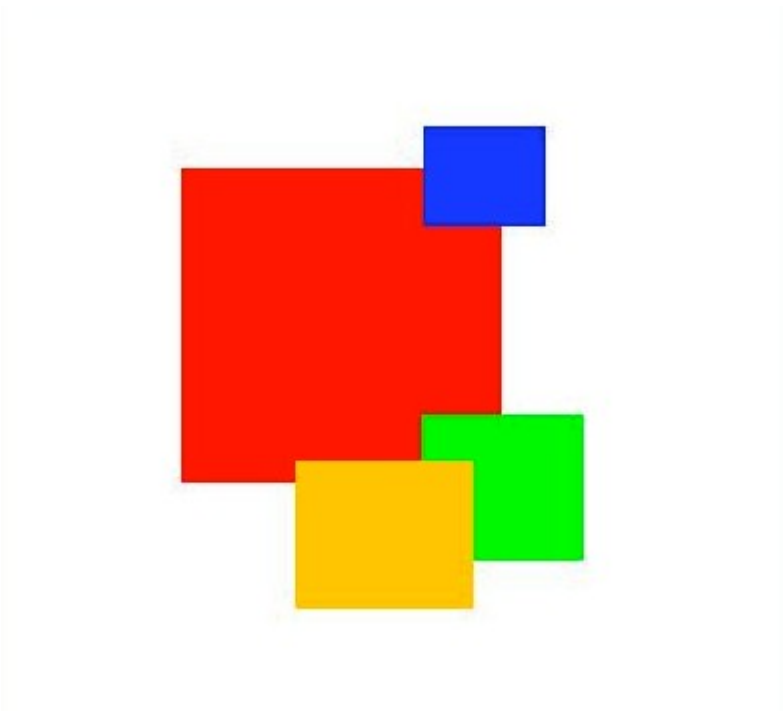
Pour mettre notre rectangle en rouge et notre cercle en vert, il suffira (moyennant les importations de la classe Color bien sûr de rajouter les lignes suivantes :

```
rect1.setFill(Color.RED);
        cerc1.setFill(Color.GREEN);
```

pour obtenir



Exercice



A vous de réaliser le groupe suivant :

Approfondissement de la classe Paint

Si vous regardez la classe abstraite Paint, vous pouvez voir que cette dernière peut être instanciée avec les classes ImagePattern (dans ce cas le remplissage est constitué d'une image) ou de deux classes de gradient LinearGradient et RadialGradient (gradient linéaire ou en cercle de couleurs).

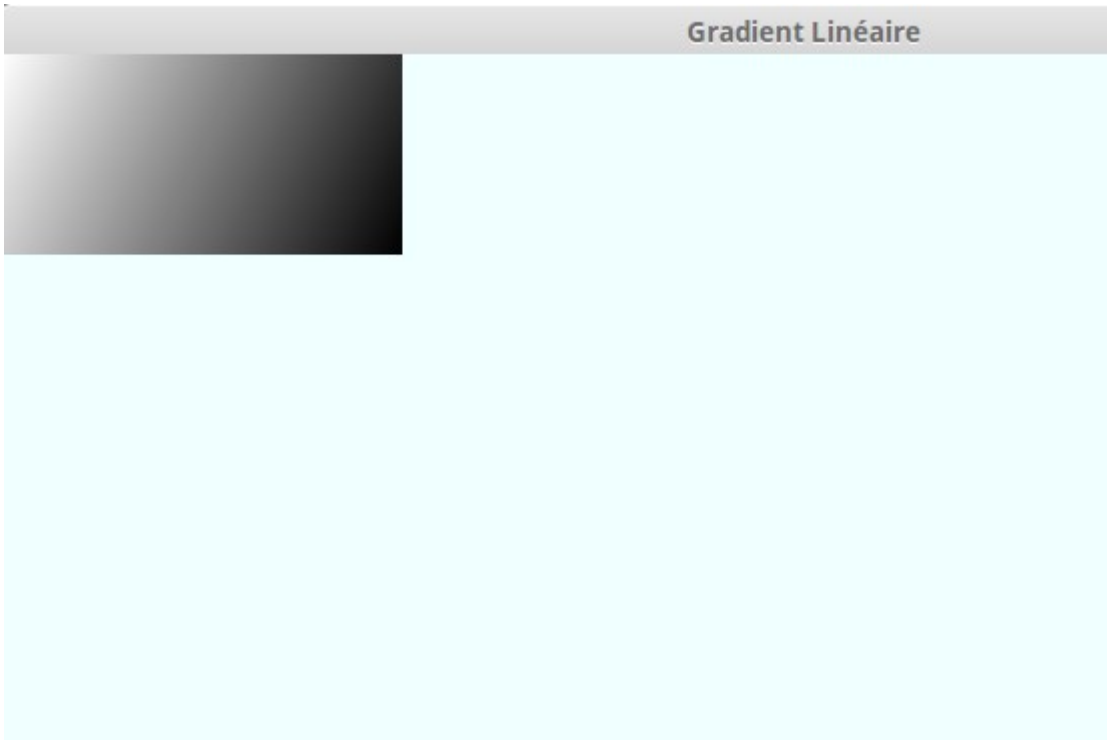
Afin de définir un gradient, il est nécessaire de définir 5 propriétés :

- indiquer le point de départ de la première couleur,
- indiquer le point d'arrivée de la couleur de fin ;
- indiquer si on utilise le coordonnées de l'écran pour les points de départ et d'arrivée ou si on travaille sur un carré de taille 1 (dans le cas les coordonnées du premier point sont (0,0) et le point final doit se situer entre (0,0) et (1,1)) ;
- indiquer si me cycle de couleur se répète dans la figure ou non ;
- définir un tableau (List) de couleurs Stop, il peut y avoir plusieurs couleurs dans votre gradient.

Voici un exemple d'utilisation du gradient linéaire dans un rectangle pour passer du noir au blanc.

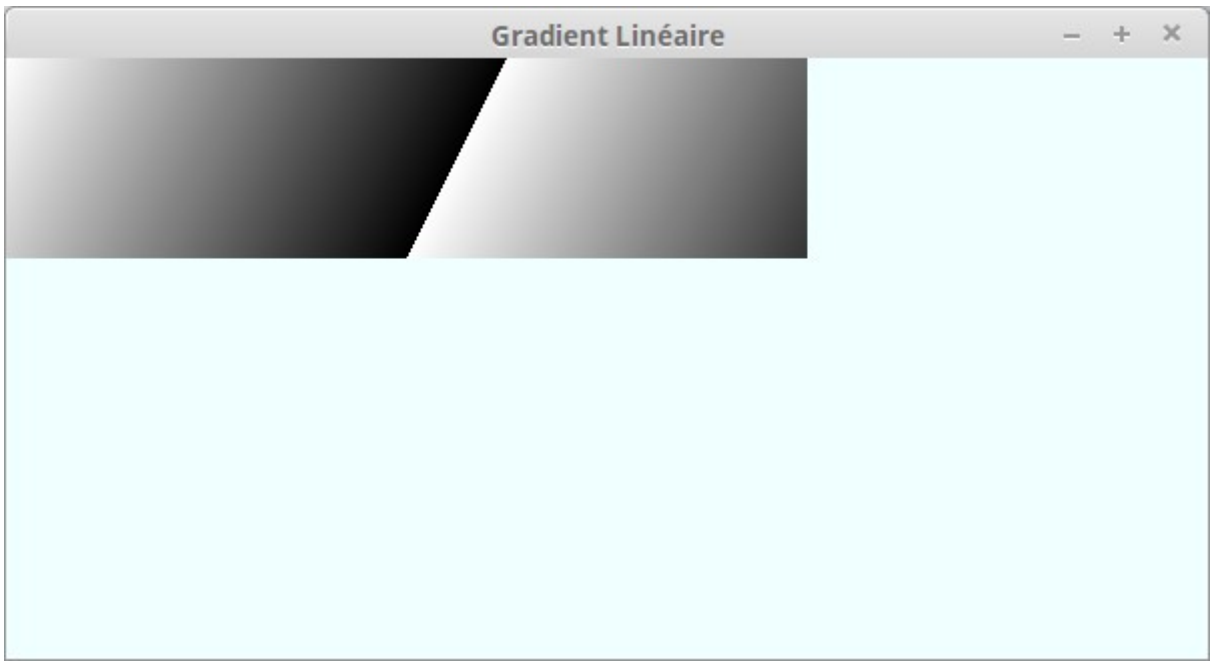
```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;

import javafx.scene.shape.Rectangle;

import javafx.scene.paint.Color;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.Stop;

import javafx.stage.Stage;

import java.util.List;
import java.util.ArrayList;

public class GradientLinéaire extends Application {
   @Override
   public void start(Stage primaryStage) throws Exception {

       //création d'une List de couleurs
       List<Stop> couleurs = new ArrayList<Stop>();
       couleurs.add(new Stop(0,Color.WHITE));
       couleurs.add(new Stop(1,Color.BLACK));

       //c création du gradient
       LinearGradient lg = new LinearGradient(0,0,200,100,
                                              false,
                                              CycleMethod.REPEAT,
                                              couleurs);

       // création du rectangle et remplissage avec le gradient
       Rectangle r1 = new Rectangle(0,0,200,100);
       r1.setFill(lg);
```

```
        //Création d'un objet de type Groupe --> Scene
        Group group = new Group();

        group.getChildren().add(r1);

        //Création de la Scene en lui passant le noeud racine
        //ainsi que la largeur et la hauteur
        Scene scene = new Scene(group ,800, 600);

        //On remplit la Scene d'une couleur
        scene.setFill(Color.AZURE);

        //Ajout d'un titre à la fenêtre --> Stage
        primaryStage.setTitle("Gradient Linéaire");

        //On ajoute la Scene au stage --> Stage
        primaryStage.setScene(scene);

        //On affiche le stage
        primaryStage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```



Comme vous pouvez le voir, notre rectangle a exactement la même taille que notre gradient linéaire (il commence en (0,0) et finit en (200,100).

Que se passe-t-il si notre rectangle est plus grand. Tout dépend de l'argument fournit au cycle. Dans le cas d'un répétition, si notre rectangle est deux fois plus grand, on obtiendra le résultat suivant :

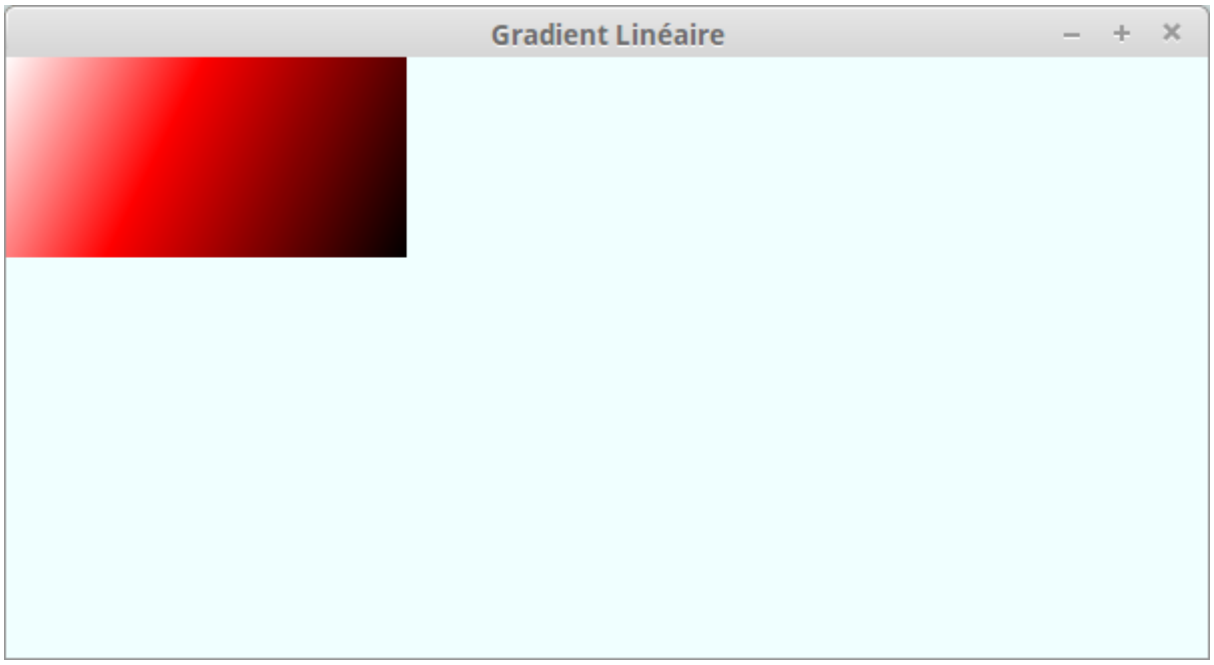


Il est bien évidemment possible d'utiliser plusieurs couleurs dans la liste « Stop ».

En utilisant le code suivant :

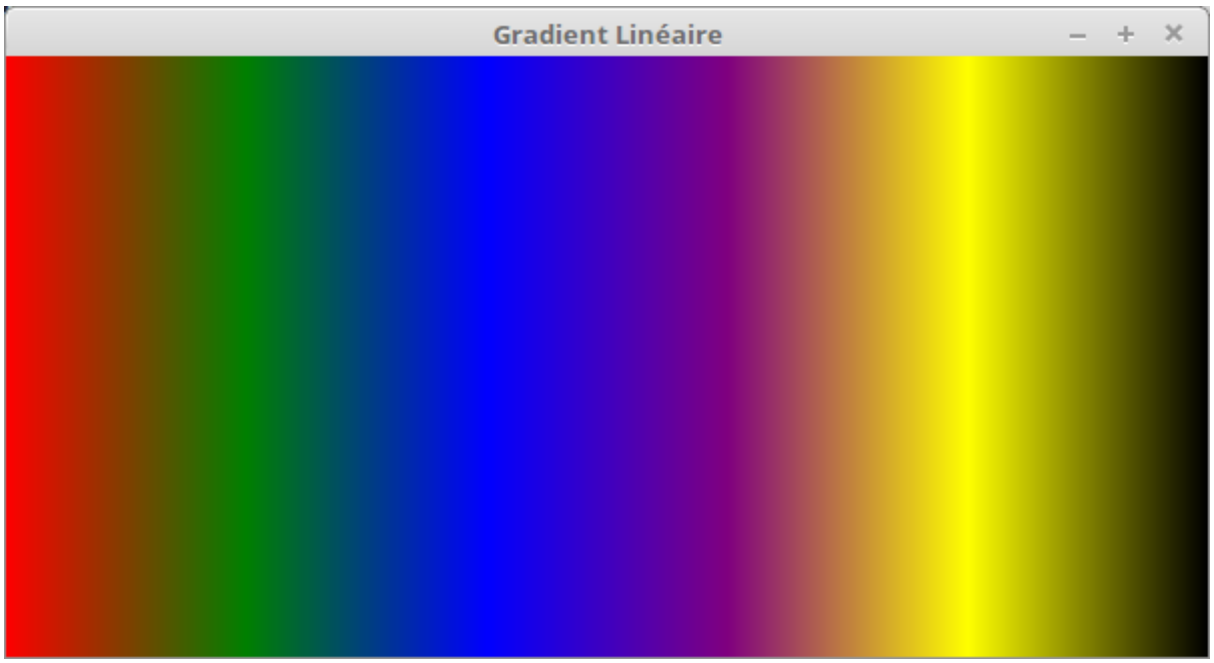
```
couleurs.add(new Stop(0,Color.WHITE));
couleurs.add(new Stop(0.4,Color.RED));
couleurs.add(new Stop(1,Color.BLACK));
```

On obtient le passage de blanc à rouge puis noir :



Exercice :

Réaliser le gradient suivant qui passe du rouge, au vert, au bleu, au violet et enfin au jaune



Exercice 2 :

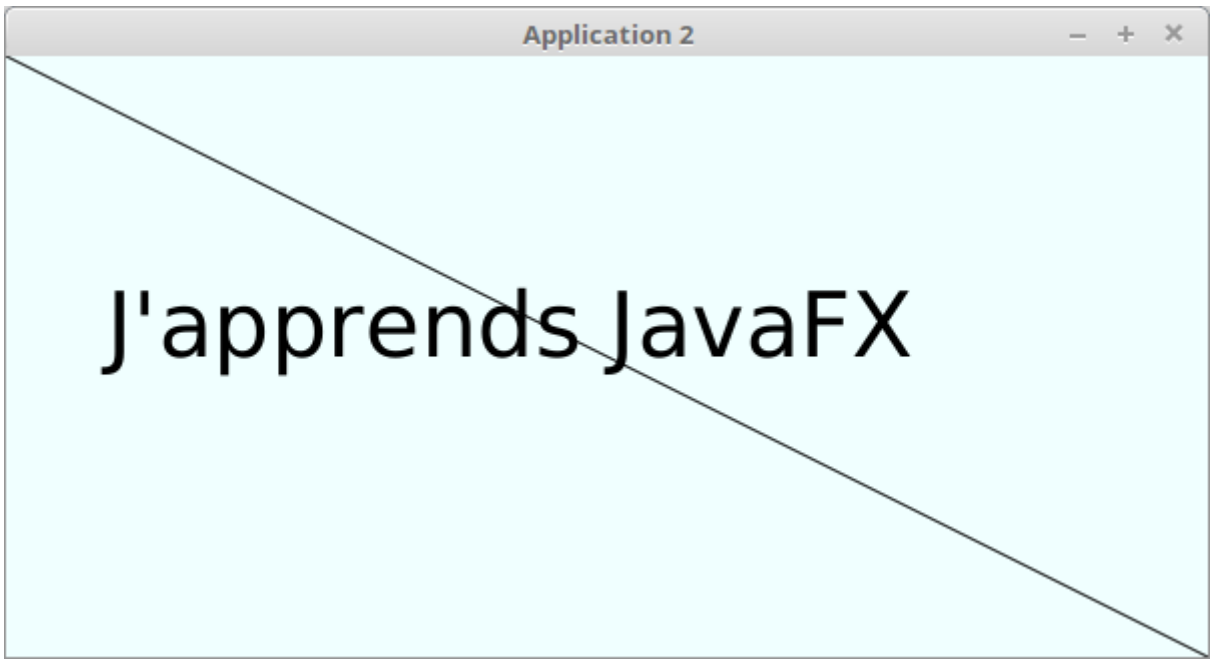
Réaliser l'exemple suivant :

Pour vous aider, j'ai réalisé un gradient dans un carré 1x1 des différents couleurs de l'arc-en-ciel qui sont RED ORANGE YELLOW GREEN BLUE INDIGO VIOLET que j'ai mis comme remplissage de la « scène ». J'ai rajouté un cercle de couleur LIGHTGREY avec un contour blanc en haut à gauche

Rajout de texte

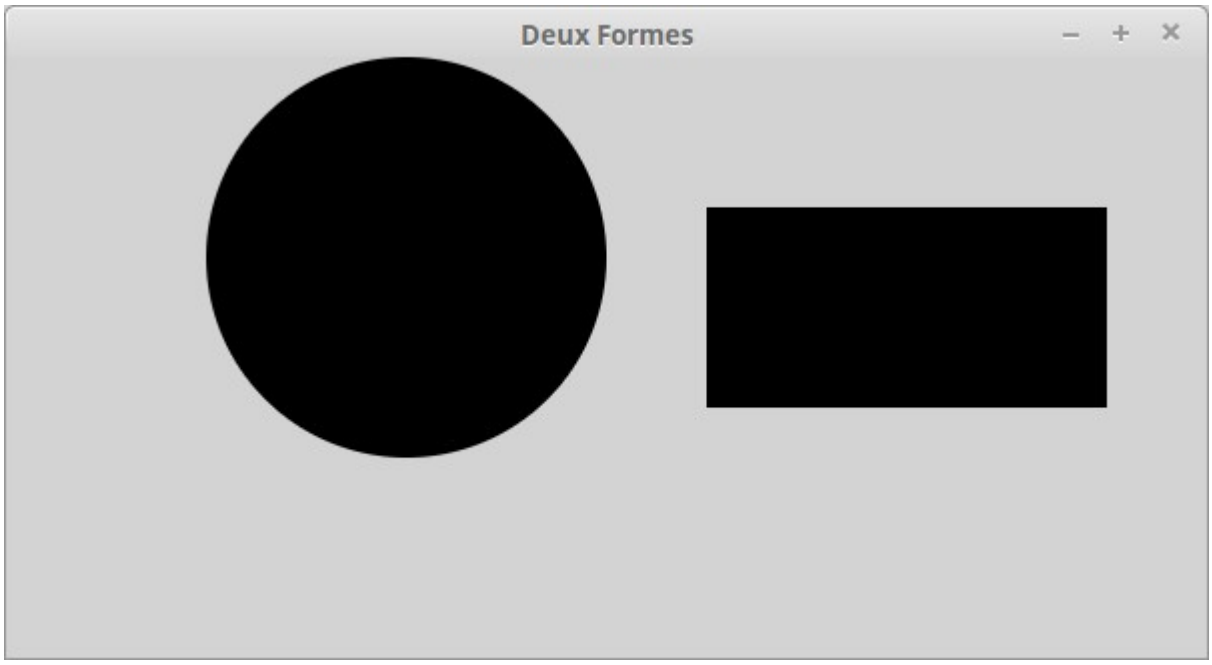
Vous commencez à comprendre le principe. Nous allons maintenant l'utiliser pour rajouter un objet de type « texte ». Il existe bien évidemment une classe Text qui permet de créer dans JavaFX des objets textuels.

Essayez d'obtenir avec les informations données la fenêtre suivante :



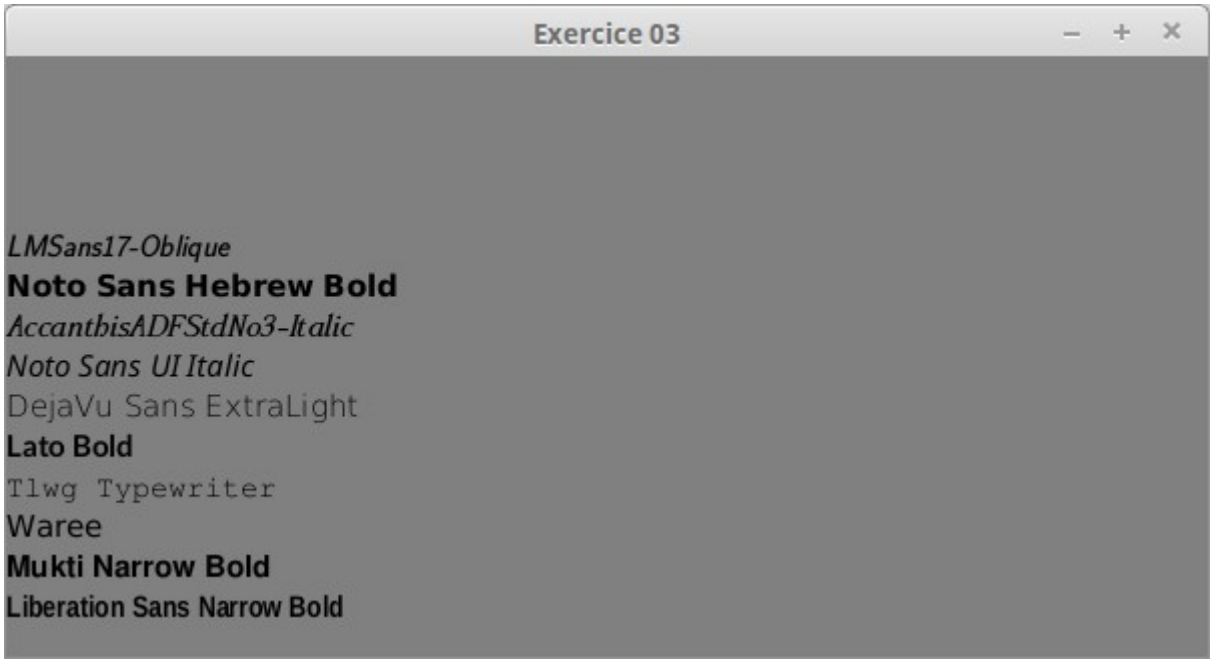
En sachant que pour la classe Font, le constructeur permet de définir la taille de la fonte (ici taille 50)

Exercice :



En utilisant les classes Font, Text, listez dans une fenêtre 10 fontes parmi les fontes disponibles sur votre machine et écrivez j’apprends JavaFX dans les différentes fontes à l’écran dans un taille 15.

Vous devriez obtenir quelque-chose comme l’écran ci-dessous :



Vous commencez un peu à saisir la philosophie de JavaFX ? Toutes les fonctions sont « simples » et harmonisées. Par exemple, pour placer du texte à l’écran, on utilisera les méthodes setX() et setY() sur un objet de type texte mais sur tout autre objet graphique un carré, une ligne…
Certaines fonctions comme l'entourage de texte sont disponibles

Et l'héritage ?

Si vous regardez attentivement, la classe Text, vous pouvez voir que celle-ci hérite de la classe Shape. Cette classe qui hérite de la classe Node est elle-même super-classe comme Circle, Path, Polygon, Rectangle…

Que peut-on faire au niveau des Node ?

Comme indiqué précédemment, la bibliothèque JavaFX a été (dans la mesure du possible) écrite dans un souci d’harmonisation et de simplification.

Il est ainsi possible de réaliser un certain nombre d’effets au niveau du nœud. Parmi les effets les plus simples, on trouve les effets :

- La translation de nœud
- La rotation de nœud
- La mise à l'échelle du nœud.

En outre, chaque nœud dans la scène définit son propre système de coordonnées

Exercice :

Réaliser la scène suivante :

Afin de vous aider, dans cet exemple, j'ai créé 50 fois un objet de type Text qui hérite donc de la classe Shape qui elle-même hérite de la classe Node ce qui implique que l'on peut appliquer à un objet de type Text toutes les méthodes de Shape et de Node.

Pour mettre le texte en couleur aléatoirement, j'ai utilisé la méthode setFill() de la classe Shape en utilisant une couleur aléatoire. Vous pouvez mettre définir une couleur à partir de ses trois composantes de rouge, de vert et de bleu en les tirant aléatoirement.

En ce qui concerne, la rotation, comme explicitée précédemment, il suffit d'utiliser la méthode adéquate dans la classe Node (la valeur de la rotation s'exprime en degrés)

Gestion des images

JavaFX permet de facilement gérer et afficher des images. La classe qui permet de charger et de manipuler des images est la classe Image. La classe qui permet d'afficher les images est la classe ImageView. Vous pouvez remarquer que la classe ImageView hérite de la classe Node, ce qui veut dire que toutes les transformations dont nous avons parlées sont valables pour cette classe.

Voici un exemple

```java
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.Scene;

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

import javafx.scene.paint.Color;



public class PremièreImage extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.setTitle("Affichage d'une image");

        // Utilisation de la classe Image pour charger une image
        Image image = new Image("mountains-1645078_1920.jpg");

        // Utilisation de la classe ImageView pour visualiser les images
        ImageView imageView = new ImageView(image);

        // Hauteur de l'image
        imageView.setFitHeight(600);

        // conserve la ratio
        imageView.setPreserveRatio(true);

        Group groupe = new Group();

        Scene scene = new Scene(groupe,800,600,Color.LIGHTGREY);

        groupe.getChildren().add(imageView);


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

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

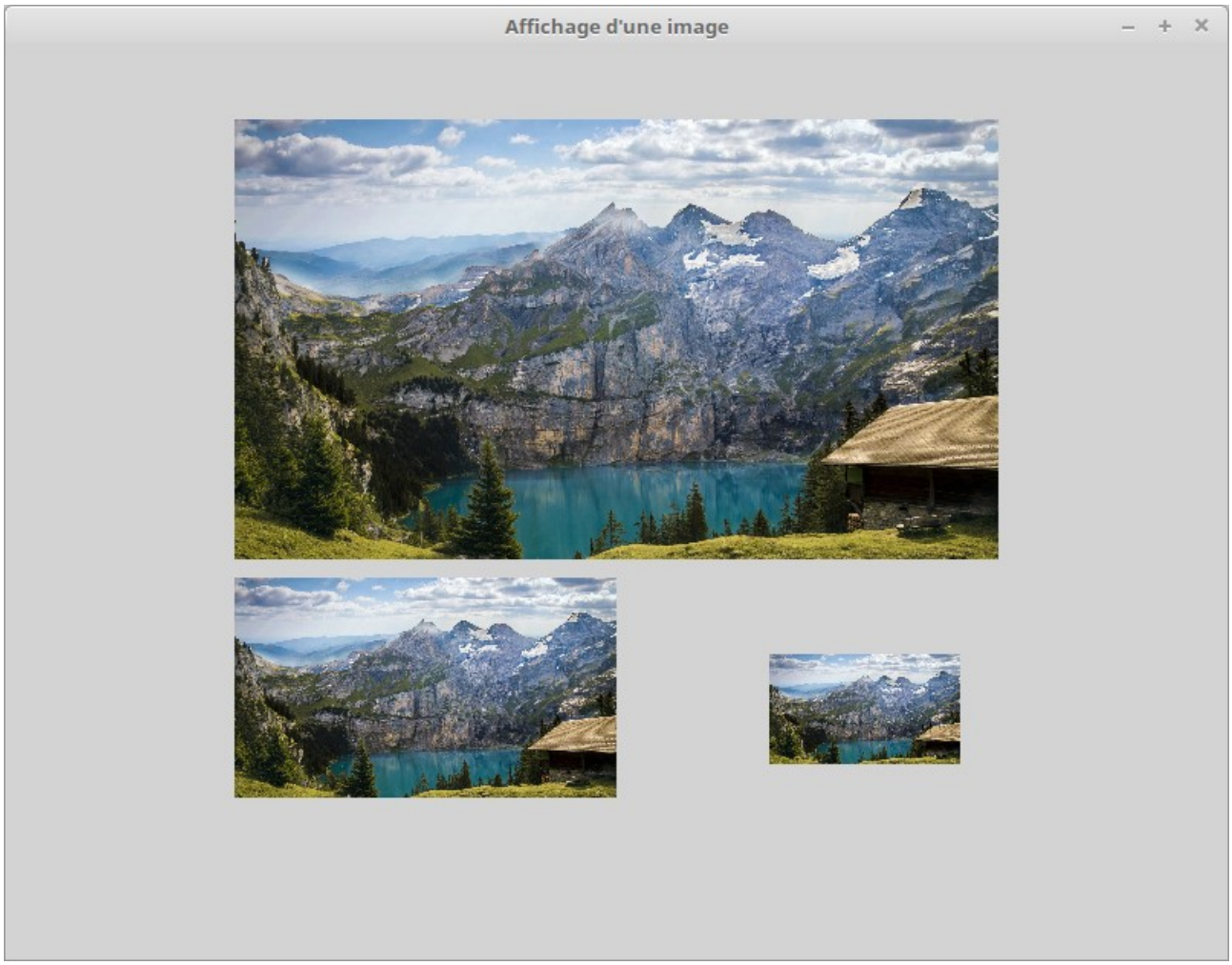

```java
}
```

qui permet d'afficher l'image suivante en conservant la proportion. Pour faire subir une rotation à l'image, il suffit comme l'image est un nœud de JavaFX de rajouter l'instruction :

```
imageView.setRotate(90);
```

Exercice :

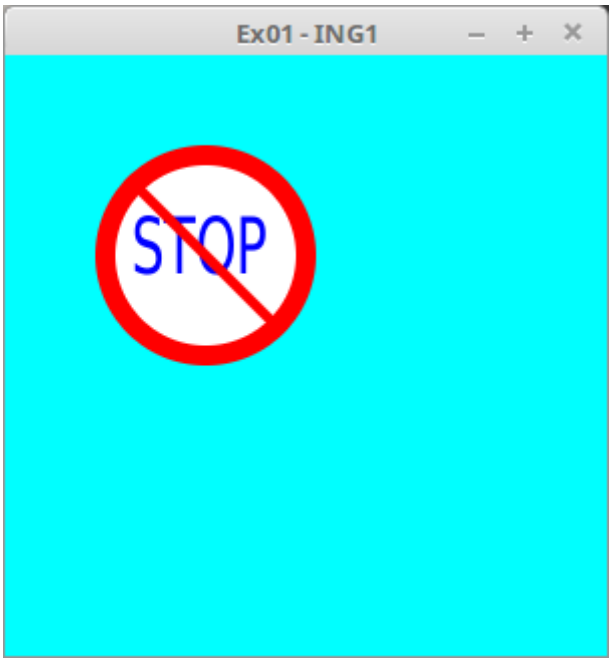
Afficher les trois images suivantes où chaque image est deux fois plus petites que la précédente



Exercices de synthèse

Exercice 1 :

Réaliser le panneau interdit de la figure ci-dessous. Pour vous aider, le panneau est formé par un cercle dont le contour est une ligne d'une certain épaisse (héritage de la classe Shape https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape.html). La barre est une ligne (de la classe Line).



Exercice 2 :

Réalisation d'un échiquier. Réaliser l'échiquier ci-dessous. Pour le réaliser, vous pouvez dessiner un grand rectangle de taille (10,10,160,160) et d'alterner des rectangles couleurs noire et blanche.