TP3

Bases de Java FX 8 : Dessinez c'est gagné.

Ceci est une nouveauté de Java 8. Il est donc impératif que vous ayez un JDK 1.8 d'installé.

Ce TP et les suivants sont basés sur plusieurs sources. Si vous êtes curieux, lisez les :

- La documentation Oracle
- Introduction à Java FX pour les jeux
- les designs Ui et Ux

Hello World

Pour tous les exercices de ce TP, on vous demandera de créer un projet Java simple (comme pour les 2 premiers TP). Nous ne créerons **pas** de projets directement *Java FX* ici.

Créez donc :

- un projet Java simple
- le package de base sera comme toujours `com.mco`
- Choisissez de créer un programme principal

Copiez/coller ensuite cette classe Main :

```
package com.mco;

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

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    public void start(Stage theStage) {
        theStage.setTitle("Hello, World!");
        theStage.show();
    }
}
```

Exécutez le code. Vous devriez voir apparaître une fenêtre avec "Hello World!" comme titre.

Particularité du code :

- la classe `Main` *hérite* de la classe `Application` définie dans la bibliothèque javafx.
- La classe `Application` définit la méthode `launch` (nous ne l'avons pas écrite) qui prépare notre application javafx.
- Une fois l'application prête, la méthode `start` (que nous avons écrite) est exécutée (par la

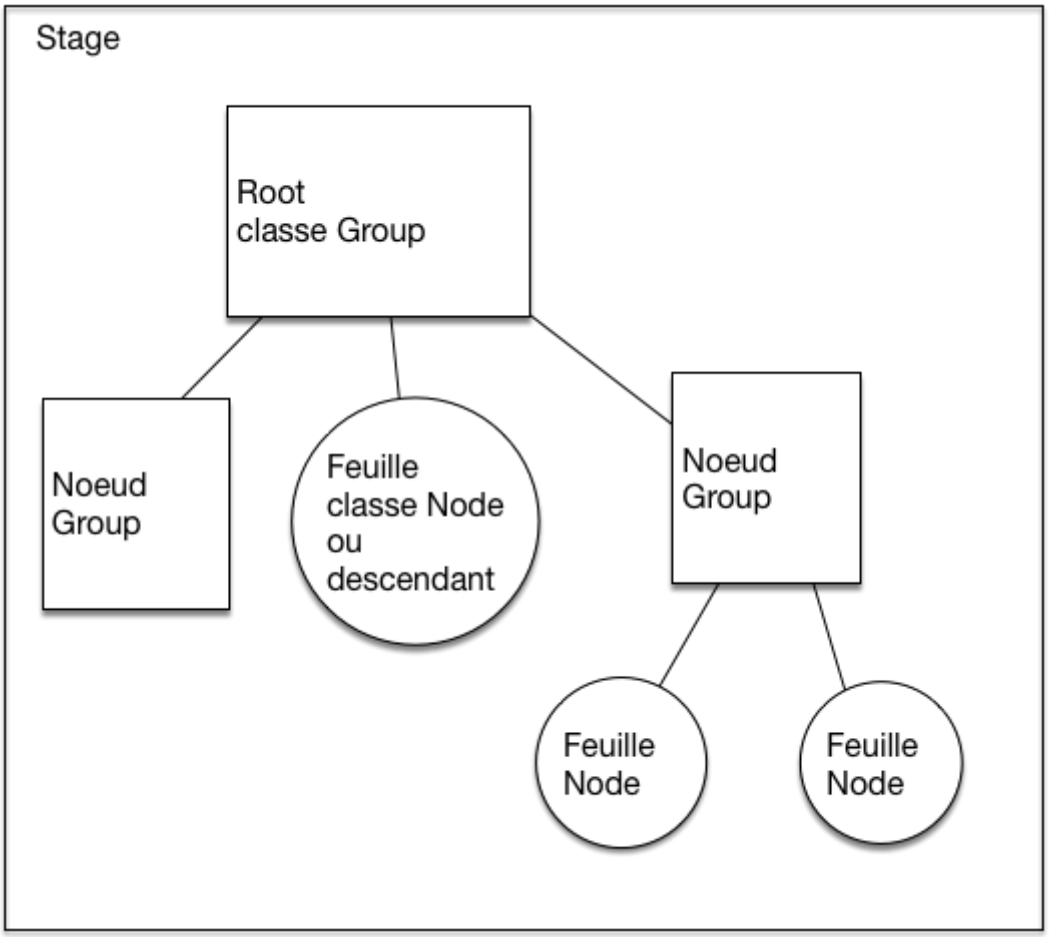
méthode `launch`) avec la fenêtre principale (de classe Stage) comme paramètre .

Cette forme d'écriture de code consistant à écrire des méthodes à l'intérieur de classes prédéfinies (que l'on sous-classe) est classique lorsque l'on utilise des frameworks web ou graphique. Il y a en effet beaucoup de choses à préparer pour créer une fenêtre et qui sont effectuées dans la classe mère (ici `Application`).

Ajoutons du contenu

Le *Stage* contient un *scene graph* contenant tout ce qui est représenté. Ce *scene graph* est un arbre dont :

- La racine est de classe Group
- les feuilles sont de la classe Node et sont ce qui est effectivement représenté Comme du Text (pour écrire) ou un Canvas pour dessiner.
- On peut ajouter un étage en ajoutant un noeud de classe Group et des feuilles à celui-ci.



Utilisation du Scene Graph

Ici nous allons :

- créer le scene graph,
- lui adjoindre son nœud root

- ajouter à cette racine une feuille qui sera du texte.
- paramétrer un peu notre texte.

```
package com.mco;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {

        launch(args);

    }

    public void start(Stage theStage) {
        theStage.setTitle("Hello, World!");

        //mise en place du Stage : création du scene graph et du noeud
racine.

        Group root = new Group(); //ce ne sera pas une feuille, donc de
classe Group.
        Scene theScene = new Scene(root); //création du scene graph
        theStage.setScene(theScene); //on le place dans le stage

        //Création d'un text
        Text text = new Text();
        text.setFont(Font.getDefault());
        text.setFill(Color.RED);

        text.setText("Hello World aussi !");
        text.setX(10);
        text.setY(50);

        //on l'ajoute à notre scene graph via son père, ici le noeud root
        root.getChildren().add(text);

        theStage.show();

    }
}
```

Spécificité du code :

- comment ajoute-t-on des fils au scene graph ?

- comment est placé le repère ?
- Pour afficher du Texte il faut :
  - un objet de la classe Text (voir la documentation Oracle)
  - une Font. Ici on a pris la police par défaut

A vous

Essayez d'autres couleurs et modifiez la position du texte.

Jouons avec le texte

Mais que fait la police ?

Les polices de caractères installées changent d'un ordinateur à l'autre. Pour les connaitre, on peut afficher à l'écran le résultat de Font.getFamilies() :

```
for (String fontName: Font.getFamilies()){
     System.out.println(fontName);
}
```

Pour créer une nouvelle police on peut alors créer une nouvelle police (`comic sans MS` est installée chez moi) :

```
Font comic = Font.font("Comic sans MS", 100);
```

Puis l'affecter à notre variable `text` :

```
text.setFont(comic);
```

A vous

Choisissez une police installée sur votre ordinateur et essayez là avec des tailles différentes.

Des dessins

Tout comme le Text, un Canvas est une feuille du scene graph. Son but est de dessiner (des boites, traits ou des images). Les possibilités sont nombreuses, voir par exemple la documentation oracle.

Carré blanc sur fond blanc

Prenons nous pour Malevitch :

```
package com.mco;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);

    }


    public void start(Stage theStage) {
        theStage.setTitle("Malevitch");

        //mise en place du Stage : création du scene graph et du noeud
racine.
        Group root = new Group(); //ce ne sera pas une feuille, donc de
classe Group.
        Scene theScene = new Scene(root); //création du scene graph
        theStage.setScene(theScene); //on le place dans le stage

        //Création d'un canvas de taille 512x512 pixels
        Canvas canvas = new Canvas(512, 512);
        GraphicsContext gc = canvas.getGraphicsContext2D(); //là où on va
dessiner

        //un carré blanc sur fond blanc
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());

        gc.setFill(Color.GHOSTWHITE);
        gc.fillRect(100, 100, 300, 300);


        //on l'ajoute à notre scene graph via son père, ici le noeud root
        root.getChildren().add(canvas);

        theStage.show();

    }
}
```

Dans un canvas on change des pixels de couleurs via un graphicsContext

A vous

Dessinez quatre ronds comme cette figure. Pour cela :

- vous pourrez utiliser une couleur semi-transparente en modifiant le chanel *alpha* qui va de 0.0 (transparent) à 1.0 (opaque).
- la méthode fillOval de la classe graphicsContext doit vous permettre de dessiner des disques.

Le Temps

Il y a plusieurs manières de gérer le temps en Java (voir ici par exemple), nous allons utiliser un timer qui s'exécutera toutes les 1/60 secondes.

Pour cela on utilise une classe Loop qui hérite de la classe AnimationTimer.

La classe AnimationTimer définit une méthode start et stop pour démarrer et arrêter le timer. Nous n'avons qu'à écrire une méthode handle qui sera exécutée environ 60 fois par seconde.

Nanosecondes écoulées

Le code suivant Crée un objet de classe Loop qui hérite de AnimationTimer. De là, après l'utilisation de sa méthode start, la méthode handle est exécutée toutes les 1/60 seconde avec comme paramètre le temps mesuré en nanosecondes.

Main.java

```
package com.mco;

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

public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Le temps");

        Loop loop = new Loop();

        loop.start();
        primaryStage.show();
    }
```

```
}
```

Loop.java

```
package com.mco;

import javafx.animation.AnimationTimer;

public class Loop extends AnimationTimer {
    @Override
    public void handle(long now) {
        System.out.println(now);
    }
}
```

Lors du premier appel à handle, la valeur du paramètre now n'est pas forcément égal à 0, ni même positif. La seule certitude est a différence entre deux appels successifs est égale au nombre de nanosecondes entre ces deux appels. C'est pourquoi on utilise quasiment toujours le delta entre 2 appels (comme vous le ferez tout de suite).

A vous

1. Plutôt que d'afficher à l'écran le temps, affichez le delta entre l'activation actuelle et l'activation précédente de la méthode `handle` (pour cela on pourra créer un attribut stockant le temps courant) : vérifiez que le temps écoulé est quasi-constant et qu'il correspond à 1/60 secondes.
2. Plutôt que d'afficher le delta à l'écran, affichez-le dans la fenêtre javafx. Pour cela :
    - on pourra passer un texte en paramètre de la création de l'objet Loop
    - on écrira son texte à chaque exécution de la méthode handle.
    - on créera la scène avec une taille par défaut en utilisant le constructeur à trois paramètres : `Scene theScene = new Scene(root, 100, 200);`

Carré

Le delta de temps (mesuré ici en nanosecondes) entre deux exécution de handle nous permet par exemple de faire bouger de façon réaliste un objet.

Le code suivant fait bouger un carré de gauche à droite.

Attention à l'initialisation pour le delta, car on ne connait pas la première valeur de now. Pour ne pas avoir de valeur aberrante, on initialise `previousTime` à la construction de l'objet avec System.nanoTime().

Main.java

```java
package com.mco;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.stage.Stage;

public class Main extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Le temps");

        Group root = new Group();
        Scene theScene = new Scene(root);
        primaryStage.setScene(theScene);

        Canvas canvas = new Canvas(512, 512);
        root.getChildren().add(canvas);

        Loop loop = new Loop(canvas);

        loop.start();
        primaryStage.show();
    }
}
```

Loop.java

```java
package com.mco;

import javafx.animation.AnimationTimer;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class Loop extends AnimationTimer {

    private Canvas canvas;

    private long previousTime;
    private double x, y;

    public Loop(Canvas canvas) {
```

```
        this.canvas = canvas;
        this.previousTime = System.nanoTime(); //évite les effet de bord à
l'initialisation de l'application

        x = 0;
        y = canvas.getHeight() / 2 - 50;
    }

    @Override
    public void handle(long now) {
        long delta = now - previousTime;
        previousTime = now;

        x += (100 * 1E-9) * delta; //avance de 100px toute les secondes

        if (x > canvas.getWidth() - 100) { //se cogne à droite de l'écran
            x = canvas.getWidth() - 100 - 5;
        }

        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.WHEAT);
        gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());

        gc.setFill(Color.MAROON);
        gc.fillRect(x, y, 100, 100);

    }
}
```

A vous

Faites osciller le carré de gauche à droite et de droite à gauche.

Une Horloge

Nous avons ici fait un exemple complet en utilisant tout ce que vous avez appris jusque là. On a dans la mesure du possible utilisé des méthodes de code courantes.

Essayez de comprendre le code :

- Les constantes sont décrites comme attributs ce qui permet de les modifier aisément (NO MAGIC NUMBERS)
- Chaque méthode est décomposée en autant de "petites fonctions" qui ne sont utilisées que là. Ceci permet de lire directement le code *via* le nom des méthodes. On les place également les unes en-dessous des autres pour faciliter la lecture. Comme ces méthodes ne sont utilisées qu'une fois on les place en `private`.
- on utilise au maximum les attributs de chaque objet pour minimiser le nombre de paramètres de chaque méthode : un objet doit être une unité fonctionnelle simple.

Main.java

```
package com.mco;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    public void start(Stage theStage) {
        theStage.setTitle("Watch Hour");

        Group root = new Group();
        Scene theScene = new Scene(root);
        theStage.setScene(theScene);

        Canvas canvas = new Canvas(512, 512);
        root.getChildren().add(canvas);

        Loop mainLoop = new Loop(canvas);
        mainLoop.start();

        theStage.show();
    }
}
```

Loop.java

```
package com.mco;

import javafx.animation.AnimationTimer;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;


public class Loop extends AnimationTimer {
    private Canvas canvas;

    final double ROTATION  = (2 * Math.PI) * 1E-9; // une rotation par
seconde
    final Color BACKGROUND_COLOR = Color.ALICEBLUE;
```

```
    final Color WATCH_HAND_COLOR  = Color.BISQUE;
    final int WATCH_HAND_LINE_WIDTH = 10;
    final double WATCH_HAND_LENGTH = 200;

    private double angle;
    private long previousTime;

    public Loop(Canvas canvas) {
        currentTime = System.nanoTime();
        this.canvas = canvas;
    }

    @Override
    public void handle(long now) {
        handleTime(now);
        drawStuff();
    }

    private void handleTime(long now) {
        changeAngle(now - previousTime);
        previousTime = now;
    }

    private void changeAngle(long delta) {
        angle += ROTATION * delta;
        angle %= 2 * Math.PI;
    }

    private void drawStuff() {
        clearCanvas();
        drawWatchHand();
    }

    private void clearCanvas() {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        gc.setFill(BACKGROUND_COLOR);
        gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
    }

    private void drawWatchHand() {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        double middleX = canvas.getWidth() / 2;
        double middleY = canvas.getHeight() / 2;

        gc.beginPath();
        gc.setLineWidth(WATCH_HAND_LINE_WIDTH);
        gc.setStroke(WATCH_HAND_COLOR);

        gc.moveTo(middleX, middleY);
```

```
        gc.lineTo(middleX + WATCH_HAND_LENGTH * Math.cos(angle),
                  middleY + WATCH_HAND_LENGTH * Math.sin(angle));

        gc.stroke();
    }
}
```

A vous

Ajouter la petite aiguille à la montre. Elle doit tourner 12 fois moins vite que la grande aiguille et être 3 fois plus petite.

Normalement, le fait d'avoir utilisé de petites fonctions bien nommée doit vous permettre d'ajouter facilement cette nouvelle aiguille.