Programmation & Algorithmique 2

TP - Séance 7

2 avril 2021

Matière visée : GUI

1 Introduction à JavaFX

L'objectif de cet exercice est de vous familiariser avec les différents concepts de JavaFX. Pour ce faire, vous trouverez sur moodle une archive contenant un projet Gradle pour JavaFX.

1.1 Compiler et exécuter

Pour compiler et exécuter le code, vous allez utiliser Gradle. Il s'agit d'un programme qui permet de facilement compiler et exécuter un programme Java tout en gérant les librairies Java à installer. Ici, Gradle va tout seul télécharger JavaFX s'il n'est pas installé sur votre machine.

Pour compiler et exécuter votre code, il suffit d'avoir un terminal ouvert sur le dossier contenant le fichier `build.gradle`. Ensuite, exécutez la commande suivante :

— `.\gradlew.bat run` si vous êtes sous Windows,
— `./gradlew run` si vous êtes sous Mac ou Linux.

Une fois le programme compilé, vous devriez voir une fenêtre qui s'affiche.

1.2 La structure de la fenêtre

Pour placer les éléments (boutons, champs texte, ...) dans une fenêtre, on utilise des layouts. Un layout est un objet dans lequel on va ajouter des éléments et qui va décider comment ceux-ci seront placés dans l'espace qu'on lui donne. Par exemple, un GridPane place ses éléments dans une grille.

Après avoir joué avec l'application, ouvrez le fichier `Main.java`. La méthode `start` est la méthode qui remplis la fenêtre avec tout ce qu'il faut. Observez bien comment les layouts (BorderPane, HBox, VBox et GridPane) sont utilisés et faites un schéma de l'organisation des éléments de la fenêtre. C'est à dire de comment chaque bouton, champ texte, ... sont organisés par ces layouts.

1.3 Le Canvas

Un Canvas est une classe spécifique à JavaFX. Mais de nombreuses librairies (Swing par exemple) graphiques ont une classe similaire. Cette classe est une zone sur laquelle on peut dessiner. Ici, on a créé une sous-classe de Canvas appelée MonCanvas. Cette classe permet de dessiner une forme (cercle ou carré) et de la faire monter ou descendre. Pour cet exercice, il vous est demandé de lire le code de MonCanvas pour comprendre son fonctionnement. Ensuite, modifiez la classe pour dessiner un carré aux angles arrondis à la place du carré.

Vous trouverez la documentation de la classe GraphicsContext à l'adresse suivante :
`https://openjfx.io/javadoc/13/javafx.graphics/javafx/scene/canvas/GraphicsContext.html`

1.4 Les handlers

La dernière étape pour l'interface graphique est de faire fonctionner les boutons. Une interface graphique fonctionne de façon évènementielle. C'est à dire qu'elle ne travaille que quand elle reçoit un évènement. Un évènement peut être le clic sur un bouton, le redimensionnement de la fenêtre, le changement d'un champ de texte, ... Pour gérer un évènement, on utilise un handler. Il s'agit d'un objet particulier qui définit ce qu'il faut faire quand on reçoit un évènement. Cet objet devra implémenter l'interface EventHandler de JavaFX afin d'avoir la méthode `handle` nécessaire pour gérer un évènement. Par exemple, on peut ajouter un handler à un bouton (via la méthode setOnAction) pour définir ce qu'on fait quand on clique sur le bouton.

Pour cet exercice, vous allez devoir lire le code de la méthode `confHandlers` de la classe `Main` pour bien comprendre ce qu'il s'y passe. Ensuite, vous devrez ajouter un bouton dans l'interface qui réinitialise le canvas quand on clique dessus.

2 Spirale

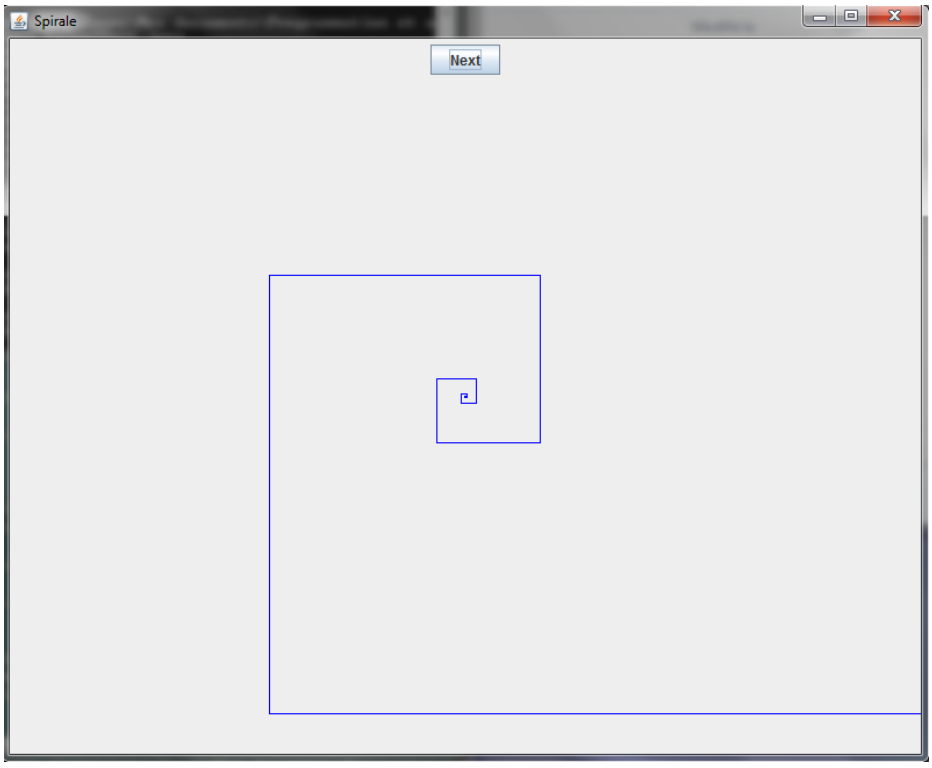

FIGURE 1 – Spirale de Fibonacci

Pour cet exercice, vous devez :

1. créer une interface `Generator` définissant l'unique méthode `public int next()`. En principe, une classe implémentant cette interface représente une "suite logique" d'entiers. La méthode `next()` permet alors d'obtenir l'entier suivant de la suite ;
2. créer une classe `Fibonacci` implémentant `Generator` représentant la suite de Fibonacci définie comme suit :

$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F(n-2) + F(n-1) & \text{sinon} \end{cases}$$

3. créer les classes nécessaires afin de générer l'affichage d'une spirale comme illustré en figure 2. Le centre du panel doit être considéré comme départ de la spirale. Le $i$ème segment utilisé

pour construire la spirale doit être de taille $F(i)$. Un bouton *NEXT* doit permettre d'afficher la spirale avec un segment supplémentaire.

4. créer ensuite une classe `PlusUn` implémentant `Generator` qui représente la suite définie par :

$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ F(n-1) + 1 & \text{sinon} \end{cases}$$

5. appliquer le point 3 avec cette suite.

3 Polygone

Créez une interface graphique laissant la possibilité à l'utilisateur d'entrer un entier n (grâce à un `TextField` par exemple) et possédant un bouton *DRAW*. Lorsque l'utilisateur appuie sur le bouton, le polygone régulier à n cotés doit être affiché et centré dans le panel. L'utilisateur peut entrer un autre entier et afficher le nouveau polygone en ré-appuyant sur le bouton sans devoir relancer le programme.

Proposez une `CheckBox` qui lorsqu'elle est cochée, permet d'afficher le cercle circonscrit au polygone.

4 Fonctions du second degré

Pour cet exercice, vous devez :

1. créer une classe `FonctionDegre2` donc les objets représentent des fonctions du second degré. Le constructeur de cette classe doit prendre en paramètres les trois coefficients a, b et c de l'équation générale :

$$ax^2 + bx + c = 0$$

   Cette classe doit contenir deux méthodes. L'une permettant d'obtenir les racines réelles de la fonction, si elles existent. L'autre prend en paramètre une valeur réelle x et retourne l'image de la fonction en x ;

2. créer une interface graphique contenant
   — une zone premettant d'entrer les valeurs de a, b et c ;
   — une zone de dessin (qui servira à l'affichage) ;
   — un bouton *DRAW*. Lorsque l'utilisateur appuie sur ce bouton, vous devez afficher :
      — les axes OX et OY ;
      — le tracé de la fonction dans l'intervalle visible par la fenêtre ; et
      — les racines de la fonction (chacune entourée par un petit cercle rouge).

5 Snake

Dans cet exercice, nous allons représenter une version simplifiée du jeu Snake (Serpent).

Dans une fenêtre, vous devez proposer une zone avec 4 boutons (un pour chaque direction), et une zone de dessin.

On représentera le serpent avec une suite de segments. Le serpent partira du milieu de la fenêtre et vers le haut.

Grâce à un `Timer`, toutes les `x` millisecondes, le serpent grandit d'une unité dans la direction actuelle. Appuyer sur un des quatre boutons permet de modifier la direction actuelle. La partie se termine si le serpent sort de la fenêtre ou s'il va sur une zone sur laquelle il est déjà présent.

On peut imaginer plusieurs versions :

1. une première version simple où le serpent ne se déplace pas vraiment, mais "grandit". Son extrémité se trouve donc toujours au milieu de la fenêtre, et la taille du serpent augmente à chaque étape.
2. une seconde version plus compliquée, où le serpent se déplace. Lorsque sa tête part dans une direction, la queue suit. La taille du serpent ne change pas. Un deuxième `Timer` permettrait d'augmenter d'une unité la taille du serpent toutes les `y` millisecondes.
3. une troisième version qui permettrait d'opposer deux joueurs simultanément. Il y aurait donc deux serpents présents dans la fenêtre (l'un rouge et l'autre bleu par exemple). Ils seraient dirigés non plus avec des boutons, mais grâce aux touches du clavier (par exemple les touches ZQSD pour le premier, et "OKLM" pour le deuxième). Si un serpent sort de la fenêtre, passe sur un endroit où il est déjà présent, ou fonce dans l'autre serpent, il a perdu.