Université Lille Nord de France
Pôle de Recherche et d'Enseignement Supérieur

ULCO Université du Littoral Côte d'Opale
Pôle de Recherche et d'Enseignement Supérieur

EIL Côte d'Opale
ÉCOLE D'INGÉNIEURS DU LITTORAL

ING 1 - POO Java année 2018 -2019

Autoformation à JavaFX

Gestion des événements

Gestion des événements en JavaFX

En JavaFX, le principe général qui est utilisé est la programmation dite événementielle. Le principe de la programmation événementielle est qu'en permanence une tâche (un petit programme) surveille l'exécution du programme et agit lorsqu'un événement survient.

Un événement représente une action qui peut être très diversifiée, cela peut-être un simple clic de souris, un déplacement de la souris, un appui sur une touche ou simultané sur plusieurs touches mais cela peut-être la réception par le système d'un événement indiquant que le processeur chauffe trop, etc...

Nous avons déjà indirectement traité le cas de certains événements sans nous en rendre compte (en fait c'est la machine virtuelle Java qui les a pris en charge), comme une division par 0 ou une saisie erronée dans le cas de l'utilisation de la classe Scanner.

Cas de JavaFX

C'est la classe Event de JavaFX qui est chargée de la gestion des événements. Plus précisément ce sont les sous-classes de la classe Event qui en ont la charge comme DragEvent, KeyEvent, MouseEvent, ScrollEvent,…

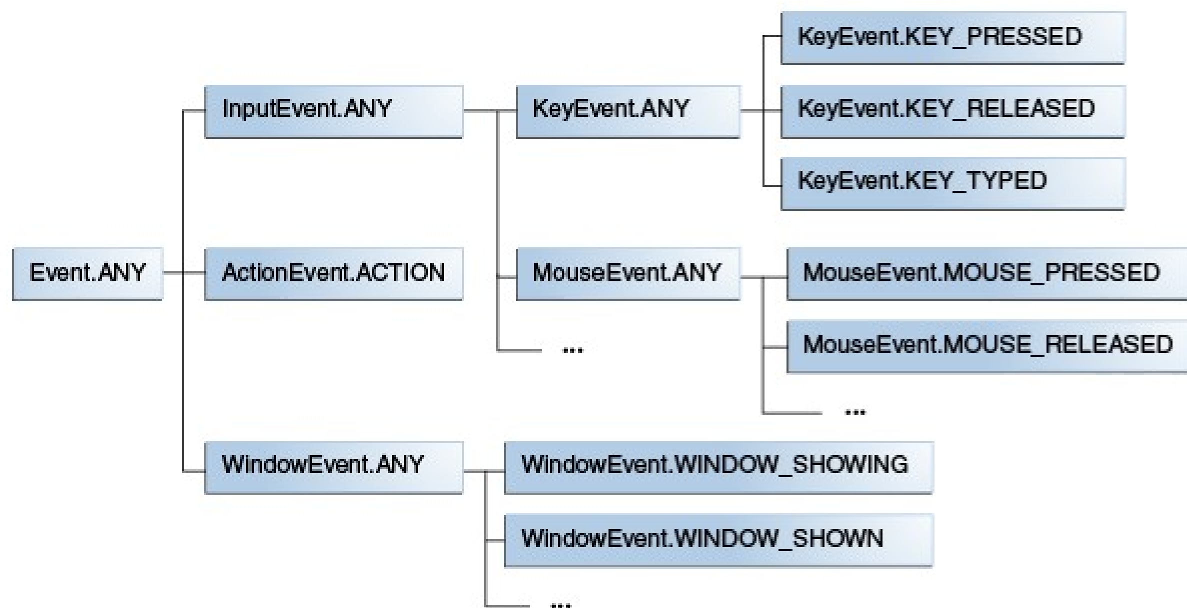
Caque événement généré possède trois informations :

Type d’événements	Source	Cible
Événement déclencheur	Origine de l’événement	Nœud sur lequel l’événement agit

Nous allons examiner plus en détails chacune des trois informations présentées ci-dessus :

Type d’événements

Tout type d’événement est un objet de classe EventType qui prend en paramètre un objet de la classe Event. Cette dernière est une classe très générique, elle est ensuite spécialisée en plusieurs sous-classes. Vous avez ci-dessous (issu de la documentation JavaFX) un aperçu de la hiérarchie des classes.



Par exemple dans le cas d'un appui sur une touche, l'événement déclencheur est KeyEvent.KEY_PRESSED, cet événement peut être récupéré au niveau de KeyEvent.ANY ou encore InputEvent.ANY,…

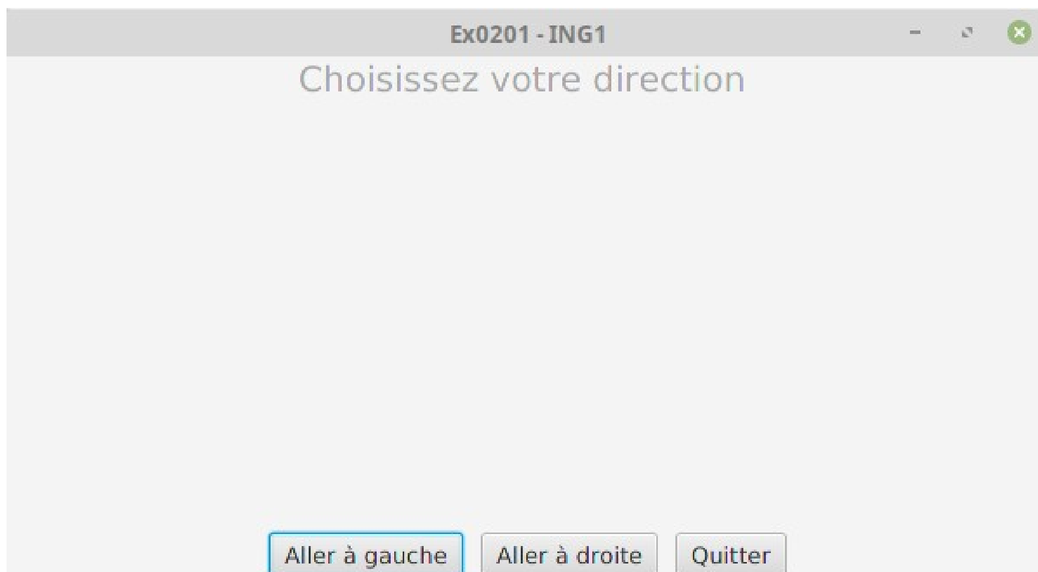
Cible

La cible d'un événement est toute classe qui implémente **l'interface** EventTarget. JavaFX a déjà implémenté cette interface dans plusieurs classes par exemple Windows, Scene ou encore Node. Avec le principe de l'héritage, cela signifie que toute classe qui hérite de la classe Node hérite des différentes propriétés de l'interface EventTarget.

Suivant l'événement réalisé, la cible peut être différente. Voici par exemples les cibles qui sont sélectionnées pour différents événements :

- Pour un événement basé sur une touche (appui mais également relâchement de la touche), la cible sera la nœud qui a le focus (*i.e.* le nœud actif) ;
- Pour un événement basé sur la souris, la cible est le nœud qui se situe à la position du curseur de souris.

Reprenons par exemple l'exemple du chapitre précédent :



Dans cet exemple, nous avons la hiérarchie javaFX suivante :

Stage

Scene

BorderPane

Label

HBox

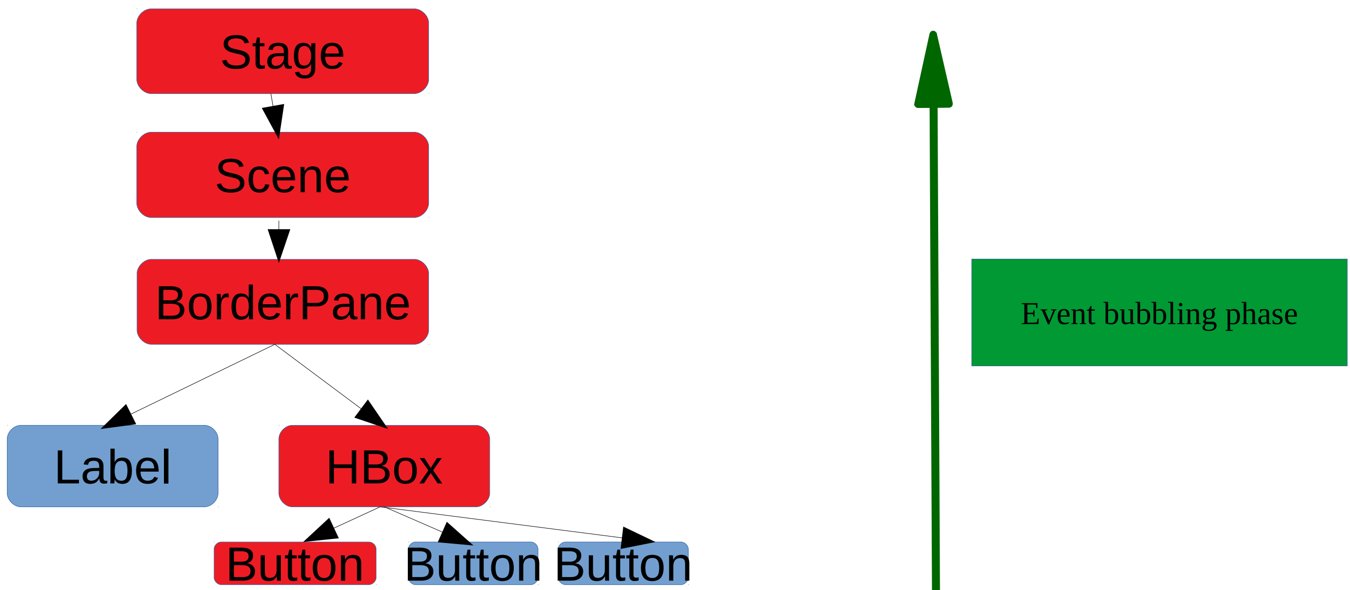
Button

Button

Button

Si vous cliquez sur le bouton gauche, c’est le bouton gauche qui sera considérée comme cible.

La chaîne des événements suivra le chemin en rouge à partir de l’objet Stage.



S'il y a ce qu'on appelle un **filtre enregistré** dans la chaîne descendant du nœud racine Stage au nœud feuille Button, ce filtre est appelé et exécuté. L'événement est ensuite transmis au nœud suivant qui à son tour s'il dispose d'un filtre peut l'exécuter.

Une fois que l'événement est arrivé au nœud cible, il remonte jusqu'à la racine et cette fois-ci (flèche verte), ce sont les gestionnaires d'événements qui sont enregistrés qui sont appelés et exécutés. Comme dans le cas de la descente d'un événement, l’exécution d'un événement ne consume pas cet événement qui est passé à son nœud père. Cette phase de remontée de l'événement est appelée Event Bubbling Phase.

En résumé, les événements peuvent être traités en JavaFX de deux manières différentes :

- Les filtres (**filters**) dans la phase descendante de la gestion de l'événement ;
- Les gestionnaires (**handlers**) dans la phase ascendant de la gestion des événements (bubbling phase).

Ce qui précède nous indique donc que pour un même nœud les filtres seront donc systématiquement exécutés avant les gestionnaires.

Comment fait-on pour créer un filtre ou un gestionnaire ?

C'est relativement aisé, les filtres et les gestionnaires sont gérés par l'interface EventHandler il y a trois possibilités :

1. Pour ajouter un filtre, on utilise la méthode **addEventFilter()** que possède tous les nœuds qui héritent de la classe Node ;
2. Pour ajouter un gestionnaire d'événements, on utilise cette fois la méthode **addEventHandler()** qui à nouveau est possédé par tous les nœuds qui héritent de la classe Node ;
3. Il est également possible d'utiliser des méthodes particulières appelées en anglais (**convenience methods**) qui permettent d'enregistrer un gestionnaire d'événements directement auprès de certains composants en utilisant le concept de propriétés de JavaFX.

Limitons nous pour le moment au deuxième cas

Les filtres et les gestionnaires d'événement doivent implémenter **l'interface** **EventHandler<T extends Event>**, ce qui revient à écrire la méthode **handle(T Event)**. T étant une sous-classe de la classe Event.

Par exemple si l'utilisateur souhaite gérer par un gestionnaire d'événement une méthode qui gère un événement de la souris, vous pourrez écrire le gestionnaire d'événements suivants :

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
   @Override
   public void handle(MouseEvent e) {
      System.out.println("Hello World");
      circle.setFill(Color.DARKSLATEBLUE);
   }
};
```

Pour l'associer un bouton par exemple, il suffira d'écrire

```
monBouton.addEventHandler(MouseEvent.MOUSE_CLICKED,eventHandler);
```

Le gestionnaire d'événements sera exécuté à chaque fois que la souris est cliquée.

Un petit exemple ?

Voici repris du chapitre précédent, la création d'un gestionnaire d'événements qui lorsque que j'appuie sur le bouton Quitter affiche un cercle rouge au centre de l'écran.

```java
import javafx.application.Application;
import javafx.collections.ObservableList;

import javafx.scene.layout.HBox;
import javafx.scene.layout.BorderPane;

import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

import javafx.scene.control.Label;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;

import javafx.scene.input.MouseEvent;

import javafx.scene.shape.Circle;

import javafx.scene.paint.Color;

import javafx.event.EventHandler;

import javafx.geometry.Pos;

import javafx.stage.Stage;



public class Ex0301 extends Application {

    private BorderPane racine = new BorderPane();
    private HBox btnPanel = new HBox(10);
    private Label labelA = new Label("Choisissez votre direction ");
    private Button btng = new Button("Aller à gauche");
    private Button btnd = new Button("Aller à droite");
    private Button btnQuit = new Button("Quitter");
    private Scene scene;
    private Circle cercle;

    // création du gestionnaire d'événements
    EventHandler<MouseEvent> eventHandler;

    public Ex0301() {
        labelA.setFont(Font.font("Arial",20));
        labelA.setTextFill(Color.DARKGRAY);
        BorderPane.setAlignment(labelA, Pos.CENTER);

        racine.setTop(labelA);

        btnPanel.getChildren().addAll(btng,btnd,btnQuit);
        btnPanel.setAlignment(Pos.CENTER);
        racine.setBottom(btnPanel);

        scene = new Scene(racine, 600, 300);
        scene.setFill(Color.OLDLACE);

        eventHandler = new EventHandler<MouseEvent>(){
                public void handle(MouseEvent e){

                    cercle = new Circle(300,150,100);
                    cercle.setFill(Color.RED);
```

```
                        racine.setCenter(cercle);

                    }
                };

            btnQuit.addEventHandler(MouseEvent.MOUSE_CLICKED,eventHandler);
        }

        public static void main(String[] args) {
            Ex0301 ex = new Ex0301();

            Application.launch(args);
        }
        @Override
        public void start (Stage stage) {

            stage.setTitle ("Ex0301 - ING1");



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

Par contre, vous pouvez remarquer que si j'appuie à nouveau sur le bouton sur le bouton Quitter, le cercle reste en place.

Exercice 1 :

À vous de modifier le code précédent de manière à ce qu'à chaque fois que vous appuyez sur le bouton Quitter, le cercle apparaisse et disparaisse alternativement.

Conseil : regardez la documentation à cette adresse
https://docs.oracle.com/javase/8/javafx/api/javafx/collections/ObservableList.html

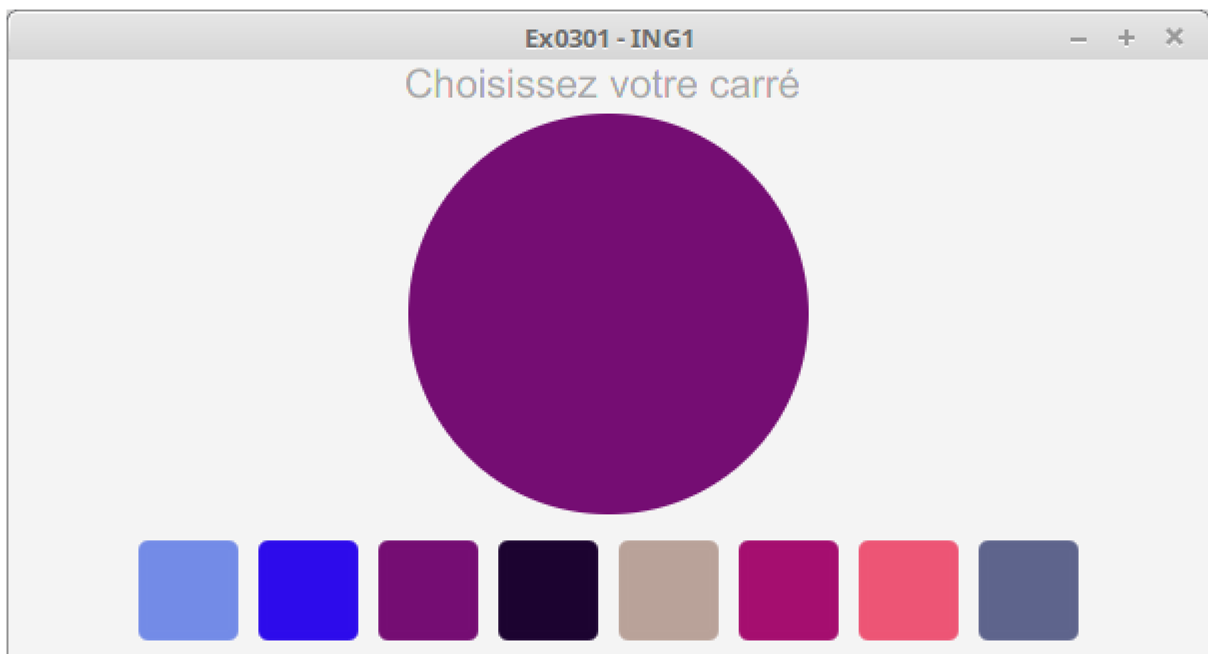
Exercice 2 :

Modifier l'exercice précédent de manière à afficher une flèche vers la gauche quand l'utilisateur appuie sur le bouton gauche, une flèche vers la droite quand l'utilisateur appuie sur le bouton droite et vous sortez du programme lorsque l'utilisateur appuie sur le bouton Quitter.

Vous n'utiliserez que des gestionnaires d'événements pour réaliser cet exercice.

Exercice 3 :

Un petit peu plus complexe et un très bon exercice de synthèse. En vous inspirant du code des exercices précédents, écrire une application javaFX qui présente dans la partie 10 carrés coloriés de manière aléatoire, l'appui sur un carré provoque l'affichage d'un cercle de la même couleur du carré.



Bien évidemment, vous devrez utiliser des gestionnaires d'événements pour provoquer l'affichage du cercle. Pour identifier, le carré source du clic sur la souris, vous pourrez utiliser la méthode **getSource()** de la classe ActionEvent.