Développement en JAVA

Java FX

JavaFX

- JavaFX a été développé originellement par Sun pour concurrence Microsoft sur le segment des RIA (Rich Internet Application)
- Depuis Java 8, Java FX est l'outil par défaut de création d'interface graphiques packagé avec le JDK
- Les applications sont multiples: média audio/video, graphisme 2D/3D, web etc…

Swing

- Swing est l'ancienne bibliothèque graphique distribuée avec Java
- Elle a été remplacée par Java FX en 2014
- Les packages java.awt.* sont ceux de Swing
- **Attention un certain nombre de classe ont les même noms entre java.awt.* et javafx.***

La classe Application

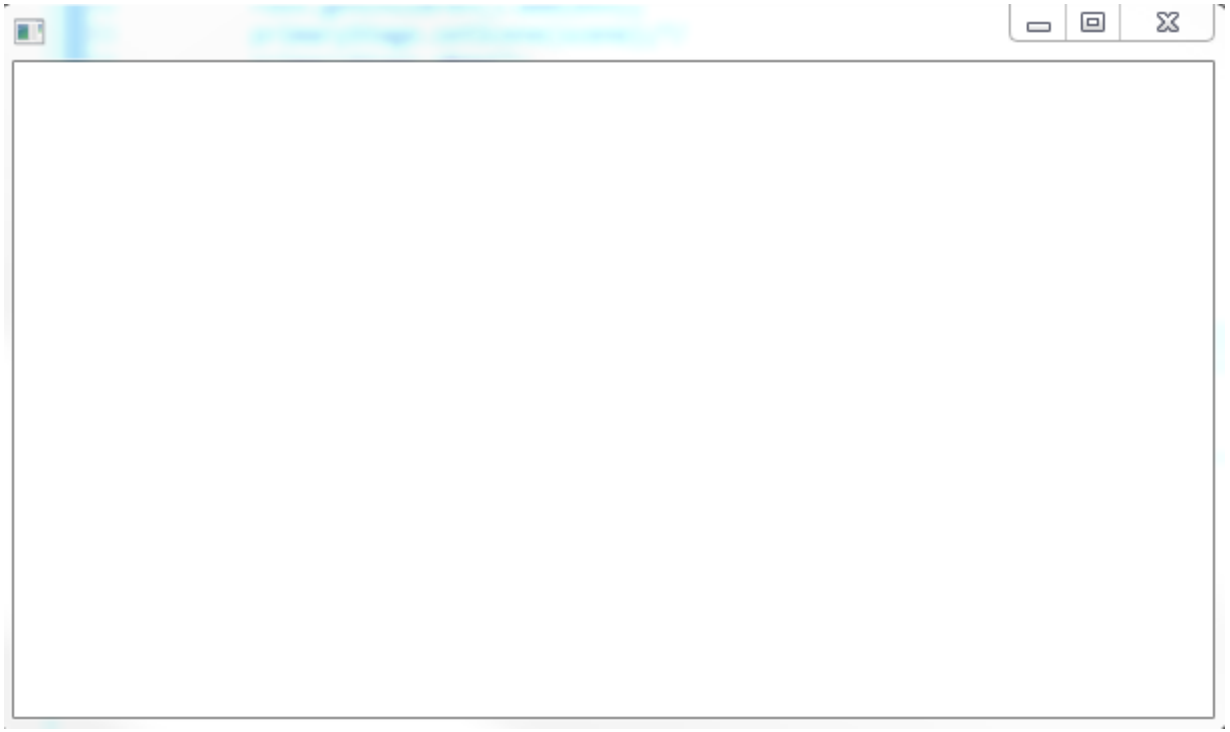
- La classe Application est la base de toute application JavaFx
- C'est une classe abstraite dont il faut implémenter la méthode `start(Stage primaryStage)`
- Cette méthode est donc le point de départ de votre application JavaFX
- La méthode statique `Application.launch(Class, args)` permet de démarrer l'interface

Exemple

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

public class Main extends Application {

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

  public void start(Stage primaryStage) throws Exception {
    // Lancement de la première fenêtre (vide)
    primaryStage.show();
  }
}
```

Exemple

- La fenêtre obtenue grâce au code précédent



Scène graphique

- JavaFX introduit le principe de scène graphique
- La classe `java.stage.Stage` représente l'endroit où s'enchaineront les différentes scènes
- On en récupère une instance lors du lancement de l'application
- A l'intérieur on peut enchainer des scènes, qui représentent les différents écrans de l'applications

La classe Stage

La classe Stage possède de nombreuse méthodes, notamment:

- La méthode `show()` qui permet d'afficher la fenêtre

- La méthode `setTitle(String)` qui permet de paramétrer le titre de la fenêtre

- La méthode `setScene(Scene scene)` qui permet de paramétrer la scène courante

La classe scène

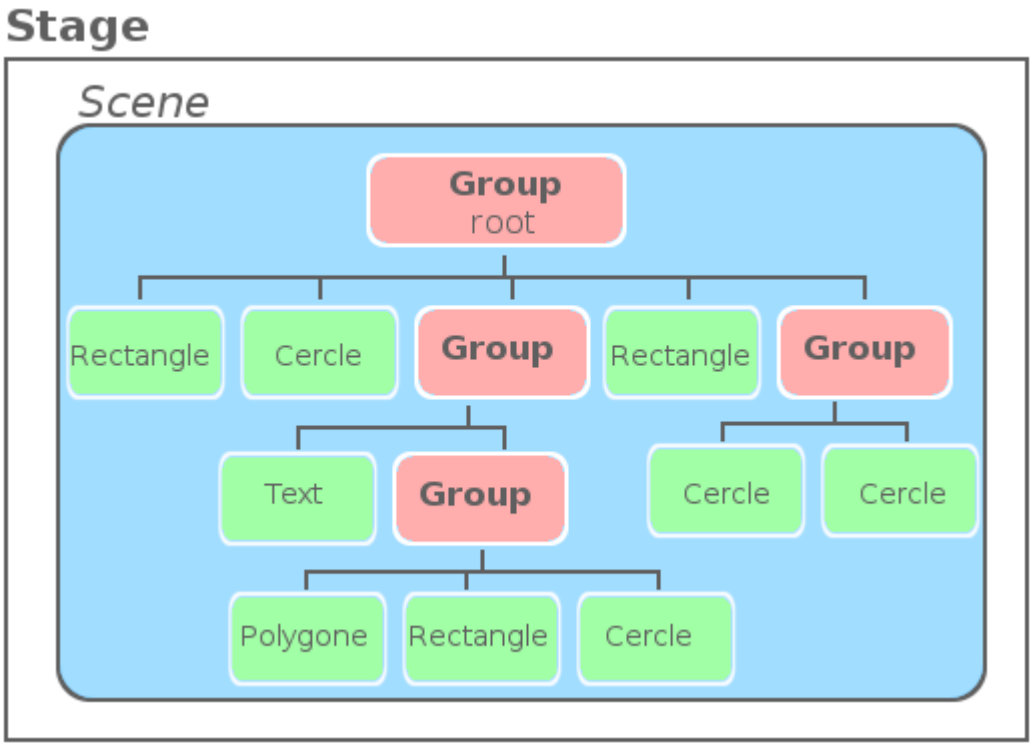
- Représente un écran donné de l'application
  - Exemple: login, écran de recherche, écran de statistiques
- Elle possède de nombreux constructeur, on peut notamment y préciser:
  - La taille de la scène
  - Le nœud à la racine de la scène
- Un scène doit être initialisée au minimum avec un élément racine

```
Scene scene = new Scene(Parent root);
```

- Le constructeur peut permettre également de préciser les dimensions de la fenêtre, sa couleur etc…

Organisation de la scène

- Les éléments de la scène sont organisés sous forme de graph qui possède des nœuds qui sont soit des groupes, soit des éléments graphiques de la scène



Les groupes

- Les groupes permettent de de regrouper plusieurs éléments de l'interface
- Un groupe peut posséder un élément ou d'autres groupes

```
Group groupe = new Group();
```

- L'ajout d'un élément à un groupe se fait à l'aide de la méthode

```
groupe.getChildren(); // renvoi une liste de node
groupe.getChildren().add(Node e);
```

Les positionnement des éléments

- La plus part des éléments peuvent être positionné à l'aide des méthodes du type setCenterX(int), setCenterY(int) etc…
- Les éléments sont affichés selon l'ordre de la liste dans laquelle ils se trouvent
- Le graph de nœud est parcouru de manière postfixé

Exemple

```
public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World");
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
        Button btn = new Button();
        btn.setLayoutX(100);
        btn.setLayoutY(80);
        btn.setText("Hello World");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            public void handle(ActionEvent event) {
                System.out.println("Hello World");
            }
        });
        root.getChildren().add(btn);
        primaryStage.setScene(scene);
        primaryStage.setVisible(true);
    }
```

L'interface Observable

- L'interface `Observable` désigne un objet dont le comportement peut être scruter par d'autres objets
- Dans une IHM on souhaite souvent scruter le comportement de certains éléments:
  - Click sur un bouton
  - Mouvement de la souris
  - Sélection d'un élément
  - etc..
- La classe `Node` définie des méthodes permettant d'écouter et de réagir aux événements

La gestion des événements

- Une méthode de gestion d'événements prend en général en paramètre un objet héritant de classe EventHandler
- Sauf cas de réutilisation, on défini cette classe de manière anonyme

```
node.setOnMouseEntered(new EventHandler<MouseEvent>() {

  @Override
  public void handle(MouseEvent event) {
    System.out.println("Youhou");
});
```

Les classes anonymes

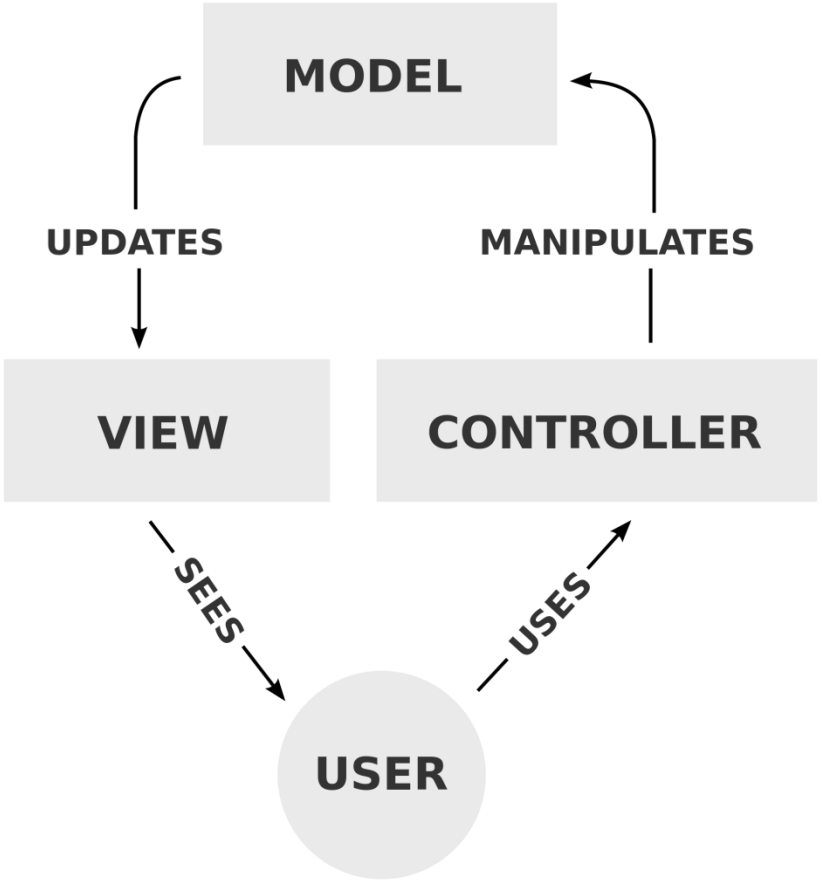
- Une classe anonyme est définie directement à l'endroit où elle est passée en paramètre et ne peut donc pas être réutiliser
- Les classes anonymes n'ont pas de constructeurs, on ne peut donc pas leur passer de paramètres
- On peut cependant accéder au contenu des variables du scope englobant, on peut parler de closure

```
Rectangle r = new Rectangle();
..
node.setOnMouseEntered(new EventHandler<MouseEvent>() {

  @Override
  public void handle(MouseEvent event) {
    System.out.println("Youhou");
    // On a accès aux variables du contexte englobant
    r.setFill(Color.AZURE); }
});
```

Pattern MVC

- Le designe pattern MVC est un patron de conception pour les interfaces que ce soit web ou client lourd

- Modèle: des classes Java
- View: des fichiers FML
- Controller: des classes Java



Les fichiers FXML

- Les fichier FXML sont des fichiers XML représentant un élément graphique
- Ils permettent de séparer la logique d'affichage des contrôles qui peuvent y être associés
- La classe FXMLLoader permet de créer un élémént un partir
- Si le chemin vers le fichier est mal positionné on obtient l'erreur suivante: Location is not set

```
FXMLLoader loader = new FXMLLoader();
// Chemin relatif à la classe courante
loader.setLocation(Main.class.getResource("../rootLayout.fxml"));
BorderPane borderPane = loader.load();
```

Scene Builder

- Scene Builder est un outil d'édition des fichiers FXML
- Il permet de réaliser très facilement des interfaces et d'accéder rapidement au catalogue des composants standards

Les contrôleurs

- Les contrôleurs sont des classes permettant de gérer les actions de l'utilisateurs

```
public class MainPanelController {

  // Ecoute d'un clic sur un bouton
  @FXML
  public void handleClick(MouseEvent event) {
    // Code exécuté lors d'un clic
  }
}
```

- L'annotation @FXML permet de désigner une méthode qui sera accessible depuis le fichier FXML

Binding vue/contrôleur

- Dans le fichier FXML une référence doit être ajoutée vers la classe contrôleur dans le tag racine

```
<BorderPane fx:controller="myapp.controller.MyController">
```

- Pour être accessible depuis le contrôleur, un nœud doit posséder un id

```
<Button  fx:id ="myButton"  text="Trame de supervision"  />
```

- Le contrôleur correspondant doit alors posséder un attribut ayant un nom et un type identique

```
public class MyController{

  @FXML private Button myButton;

  @FXML
  public void handleClick(MouseEvent event) {
    myButton.setText("clicked");
  }
}
```

Les layouts

- Les layouts du package `javafx.scene.layout` sont des structures graphiques qui permettent d'organiser vos composants d'une manière ou d'une autre:
  - BorderPane: division de la zone en cinq partie (top, down, right, left et center)
  - Hbox: alignement horizontal
  - Vbox: alignement vertical
  - GridPane: la zone est divisée en une grille ligne/colonne
  - AnchorPane: l'élément est fixé à un des bords de la zone
  - Etc…
- Les layouts sont des nœuds qui héritent de la classe Group
  - On peut donc leur ajouter d'autres nœuds

Les formes

- Le package `javafx.scene.shape` contient de nombreuses classes permettant de rendre des formes de base: Circle, Rectangle, Line, Polygon…

```
Circle circle = new Circle(100);
circle.setCenterX(200); // Position horizontale
circle.setCenterY(200); // Position verticale
circle.setFill(Color.AZURE); // Couleur de remplissage
circle.setStroke(Color.BLACK); // Couleur du contour
```

Les images

- Les classes `javafx.scene.image.Image` et `javafx.scene.image.ImageView` permettent de charger et d'afficher des images facilement

```
ImageView mon_imageview = new ImageView(new
Image("../image.jpeg"));
groupe.getChildren().add(mon_imageview);
```

Les contrôles

- Les contrôles permettent de proposer des actions standards à l'utilisateur
- De nombreux contrôles classiques sont disponibles dans le package `javafx.scene.control`:
  - Button
  - Label
  - Radio
  - TextField
  - PasswordField
  - etc.
- De nombreux exemples: https://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm

Créer ses propres nœuds

- On peut parfois avoir besoin de créer ses propres nœuds, par exemple pour pouvoir réutiliser des morceaux d'interface à plusieurs endroits
- Un nœud doit implémenter la classe javafx.scene.Parent

```
public class MonNoeud extends Parent{
  public MonNoeud(){
    //construction du nœud spécifique
    Rectangle r = new Rectangle();
   //comme pour un
    this.getChildren.add(r);
  }
}
```

CSS

- La personnalisation des interfaces peut se faire de manière très simple grâce aux feuille de style CSS
- Une feuille CSS se paramètre au niveau d'une scène

```
Scene scene = new Scene(new Group(), 500, 400);
scene.getStylesheets().add("path/stylesheet.css");
```

- Ou dans le fichier FXML

```
<Pane prefHeight="400.0" prefWidth="600.0"
stylesheets="@theme.css" …>
```