Université de Limoges

R2.02

JavaFX™

Notions avancées (ter)

Objectifs du cours

- Rappels…
- Un peu d’animation
- L’outil Scene Builder
- Gestion multi-fenêtres
- JavaFX dans un projet Maven

Rappels

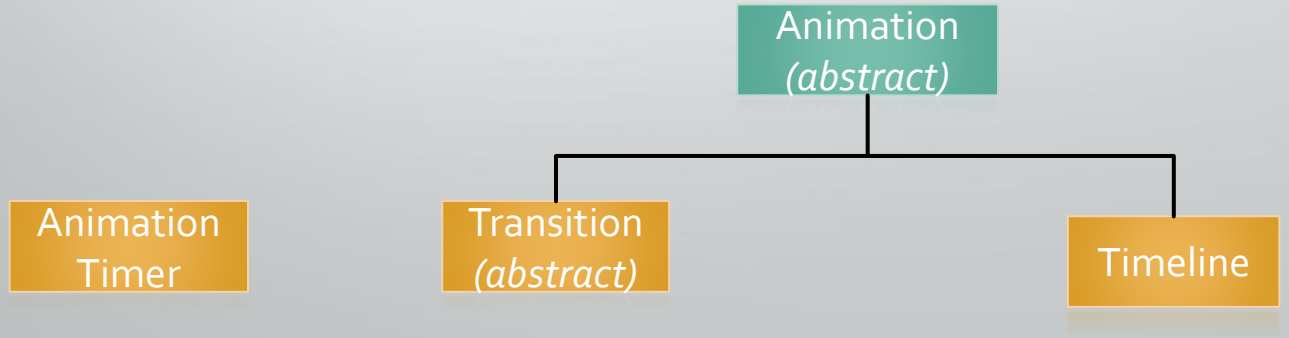
- Propriétés JavaFX et binding
- ObservableList
- ListView
- Le glisser/déposer (Drag-and-Drop) :
  - La notion de Serializable
  - Initier un DnD
  - Transfert de données
  - Évènements lors d'un DnD
- Les styles CSS

Un peu d'animation

Définition : illusion de mouvement provoquée par une suite d'images.

- Comparaison avec les flipbooks : chaque page représente une image (ou **frame**) qui est affichée selon une chronologie (**timeline**) pendant une période de temps (**duration**).

- JavaFX permet d'**animer** des objets et de réaliser des effets complexes avec peu de code.
  - Classe abstraite **Animation**
  - API d'animation → package javafx.animation.*



La classe AnimationTimer

- Permet de créer un timer (une minuterie) qui pourra être :
  - activé : méthode ***start()***
  - désactivé : méthode ***stop()***
- Quand il est activé : il pourra déclencher un traitement à chaque frame (environ 60 par seconde).
- En héritant de AnimationTimer : notre classe doit redéfinir la méthode

```
public abstract void handle(long now)
```

Le paramètre "*now*" représente le timestamp de la frame courante en nanosecondes.

Cette valeur sera identique pour toutes les AnimationTimer actifs durant une même frame.

- Exemple :

```
AnimationTimer timer = new AnimationTimer() {
    @Override
    public void handle(long now) {
        // action a chaque frame
    }
};
timer.start();
```

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/AnimationTimer.html

L' API Animation

- **Animation** : classe abstraite, fournit les fonctionnalités de base permettant de créer des animations en JavaFX
- Quelques actions possibles :
  - Jouer, mettre en pause ou arrêter : méthodes ***play( )***, ***playFromStart()***, ***pause()***, ***stop( )*** et propriété ***status***
  - Répéter un certain nombre de cycles : propriété ***cycleCount*** (infinie *Timeline.INDEFINITE* → l'animation sera cyclique)
  - Direction et vitesse variables : propriété ***rate*** et flag ***autoreverse***
  - méthode ***setOnFinished(***`EventHandler<ActionEvent>`***)*** → permet d'associer un écouteur d'évènement qui sera déclenché à la fin de l'animation

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Animation.html

Transition

Définition : *passage d'un état à un autre, en général lent et graduel.*

Synonyme : *changement, évolution.*

- **Transition** : classe abstraite permettant de réaliser des animations basées sur une transition. Une classe fille doit :
  - définir la durée d'un cycle **Animation.*setCycleDuration***(`javafx.util.Duration`)
  - implémenter la méthode ***interpolate(***`double`***)***
- Une transition :
  - porte sur un élément graphique
  - change une (ou plusieurs) propriété(s) au sens JavaFX d'une valeur d'origine vers une valeur cible →
  - s'appuie sur une interpolation (**Interpolator**)
  - l'animation peut être répétée, jouée en sens inverse, etc…
  - est effective sur une durée (**Duration**)

https://docs.oracle.com/javase/8/javafx/api/javafx/util/Duration.html

Quelques exemples de Transitions

```
Rectangle rect = new Rectangle (100, 40, 100, 100);
rect.setArcHeight(50);
rect.setArcWidth(50);
rect.setFill(Color.VIOLET);
```

- Translation :

```
TranslateTransition tt = new TranslateTransition(Duration.millis(2000), rect);
tt.setByX(200f);
tt.setCycleCount(4);
tt.play();
```

- Rotation :

```
RotateTransition rt = new RotateTransition(Duration.millis(3000), rect);
rt.setByAngle(180);
rt.setCycleCount(4);
rt.play();
```

- Disparition – apparition :

```
FadeTransition ft = new FadeTransition(Duration.millis(3000), rect);
ft.setFromValue(1.0);
ft.setToValue(0.3);
ft.setCycleCount(4);
ft.setAutoReverse(true);
ft.play();
```

- Ecouteur d'évènement à la fin de l'animation :

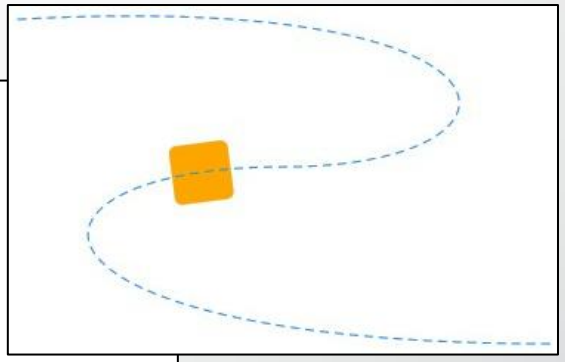
```
ft.setOnFinished(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent evt) {
        // TODO

    }
});
```

Autre exemple : le PathTransition

- Cette transition créé une transition selon un chemin ou **Path**.
- Ce chemin peut être construit à partir d'un ou plusieurs **PathElement** :

  ArcTo, **ClosePath**, **CubicCurveTo**, **HLineTo**, **LineTo**, **MoveTo**, **QuadCurveTo**, **VLineTo**

- Exemple :



```
final Rectangle rectPath = new Rectangle (0, 0, 40, 40);
rectPath.setArcHeight(10);
rectPath.setArcWidth(10);
rectPath.setFill(Color.ORANGE);

// Creation du chemin
Path path = new Path();
// MoveTo permet de se rendre a des coordonnees (X,Y) specifiques
path.getElements().add(new MoveTo(20,20));
// CubicCurveTo permet de creer une courbe de Bezier definie par 3 points
path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120));
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));
// Creation de la PathTransition
PathTransition pathTransition = new PathTransition();
pathTransition.setDuration(Duration.millis(4000));
pathTransition.setPath(path);
pathTransition.setNode(rectPath);
pathTransition.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
pathTransition.setCycleCount(Timeline.INDEFINITE);
pathTransition.setAutoReverse(true);
pathTransition.play();
```

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/PathTransition.html

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/PathElement.html

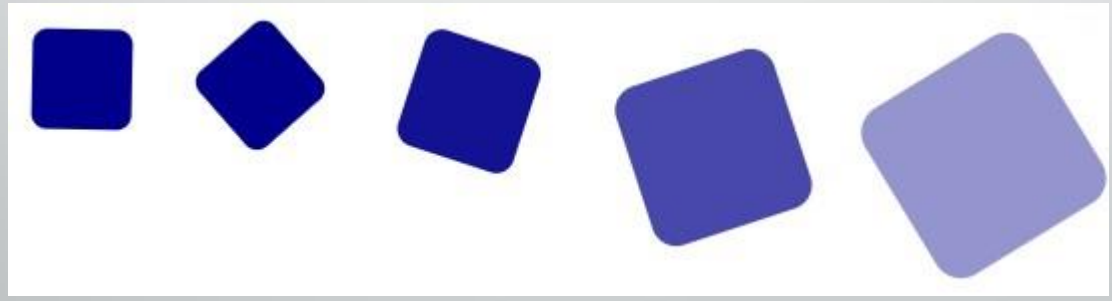
Séquence ou série de transitions

- On peut regrouper des transitions pour les exécuter en parallèle ou bien de manière séquentielle
- Transitions séquentielles :

```
SequentialTransition seqT = new SequentialTransition(rect, ft, tt, rt);
//seqT.getChildren().addAll(ft, tt, rt);
seqT.play();
```

- Transitions en parallèle :

```
ParallelTransition pt = new ParallelTransition(rect, ft, tt, rt);
//pt.getChildren().addAll(ft, tt, rt);
pt.play();
```



https://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm

Liste des Transitions JavaFX

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Transition.html	
TranslateTransition	Effet de translation qui s'étend sur une durée
RotateTransition	Effet de rotation qui s'étend sur une durée
PathTransition	Animation d'un élément le long d'un chemin ou Path (propriétés impactées translateX, translateY et rotate)
ScaleTransition	Effet de redimensionnement d'un élément avec un facteur d'échelle sur trois dimensions (scaleX, scaleY, scaleZ)
FadeTransition	Effet de fondu (propriété impactée opacity)
StrokeTransition	Effet modifiant la couleur de trait d'une forme
FillTransition	Effet modifiant la couleur de remplissage d'une forme
PauseTransition	Simule un temps de pause (animation qui ne fait rien)
SequentialTransition	Exécute séquentiellement une liste d'animations
ParallelTransition	Exécute une liste d'animations en parallèle

Transition : pour finir

- **Une Transition** ne modifie pas la valeur de la Property JavaFX

- L'image est modifiée, mais pas forcément les attributs de l'objet sous-jacent.

Exemple : une translation ne va pas modifier les coordonnées (*X*, *Y*) mais uniquement les valeurs de *translateX* et *translateY*.

- Si besoin :
  - vous pouvez les modifier vous-même en fin de transition (via la méthode ***setOnFinished*** évoquée précédemment)…
  - ou bien passer par une animation de type **Timeline**.

L'animation de type Timeline

- La classe **Timeline** permet l'animation d'un élément :
  - Basée sur une **property** **JavaFX** (telle que la taille, la position, la couleur, etc) ; elle devra être modifiable (**WritableValue**).
  - La classe **KeyValue** contiendra la nouvelle valeur modifiée de cette property, pour un instant donné de l'animation**.**
  - La classe **KeyFrame** contiendra une liste de **KeyValue** ainsi qu'une durée (**Duration**) : c'est la liste des valeurs successives que devra prendre la property, sur une période de temps donnée.
- La **Timeline** rassemble des **KeyFrame** qui seront jouées séquentiellement.
- Elle hérite de la classe **Animation** : elle peut donc se manipuler comme telle !

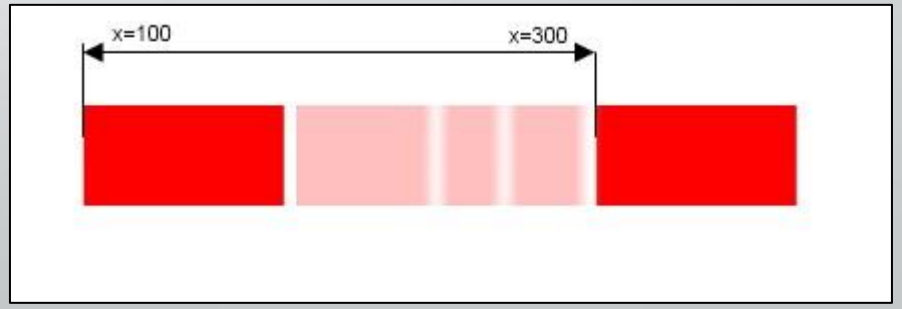
https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Timeline.html

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/KeyFrame.html

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/KeyValue.html

Exemple d'utilisation de la Timeline

```
// creation d'un Rectangle rouge (position X=100)
final Rectangle rectangle = new Rectangle(100, 50, 100, 50);
rectangle.setFill(Color.RED);

// definition de la valeur cible de la Property (position X=300)
final KeyValue kv = new KeyValue(rectangle.xProperty(), 300);

// creation d'un KeyFrame d'une duree de 500ms contenant la KeyValue
final KeyFrame kf = new KeyFrame(Duration.millis(500), kv);

// creation de la Timeline
final Timeline timeline = new Timeline();
// se repete a l'infini
timeline.setCycleCount(Timeline.INDEFINITE);
// inverse la direction durant les differents cycles
timeline.setAutoReverse(true);
// ajout de la KeyFrame dans la Timeline
timeline.getKeyFrames().add(kf);
// on joue l'animation
timeline.play();
```



Timeline : notion d'interpolation

Définition : *reconstitution, en général approximative, d'une grandeur continue à partir d'un ensemble discret de valeurs intermédiaires de cette grandeur.*

- Les **KeyValue** indiquent les différentes valeurs de la **property** sur laquelle se base l'animation **Timeline**.
- Une interpolation définit les valeurs intermédiaires entre deux valeurs (successives) : la valeur source et la valeur de destination.
- Ce calcul est pris en charge par un objet de type **Interpolator**.
- Les différents types d'interpolateurs fournis par JavaFX :
  - LINEAR : interpolation linéaire (par défaut)
  - DISCRETE : interpolation à temps discret
  - EASE_IN : l'animation démarre lentement puis accélère doucement
  - EASE_OUT : l'animation ralentit doucement vers la fin
  - EASE_BOTH : l'animation commence lentement, puis accélère et ralentit à nouveau vers la fin, le tout de manière fluide
- On peut aussi écrire notre propre implémentation d'un Interpolator.

https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Interpolator.html

Deux exemples d'interpolation

Ex1 : un objet KeyValue permettant de rendre progressivement invisible un rectangle → en interpolant son opacité de la valeur 1 à la valeur 0.

```
Rectangle rectangle = new Rectangle(0, 0, 50, 50);
KeyValue keyValue = new KeyValue(rectangle.opacityProperty(), 0);
```

L'objet KeyValue définit uniquement les valeurs source et destination, mais ne spécifie pas d'Interpolator (interpolation linéaire par défaut).

Ex2 : un objet KeyValue permettant de déplacer un rectangle de gauche à droite, de 100 pixels, avec un Interpolator de type EASE_OUT

```
Rectangle rectangle = new Rectangle(0, 0, 50, 50);
KeyValue keyValue = new KeyValue(rectangle.xProperty(), 100, Interpolator.EASE_OUT);
```

L'Interpolator.EASE_OUT va ralentir le mouvement de l'animation juste avant d'atteindre la valeur de destination (x=100).

Autres possibilités : les effets visuels

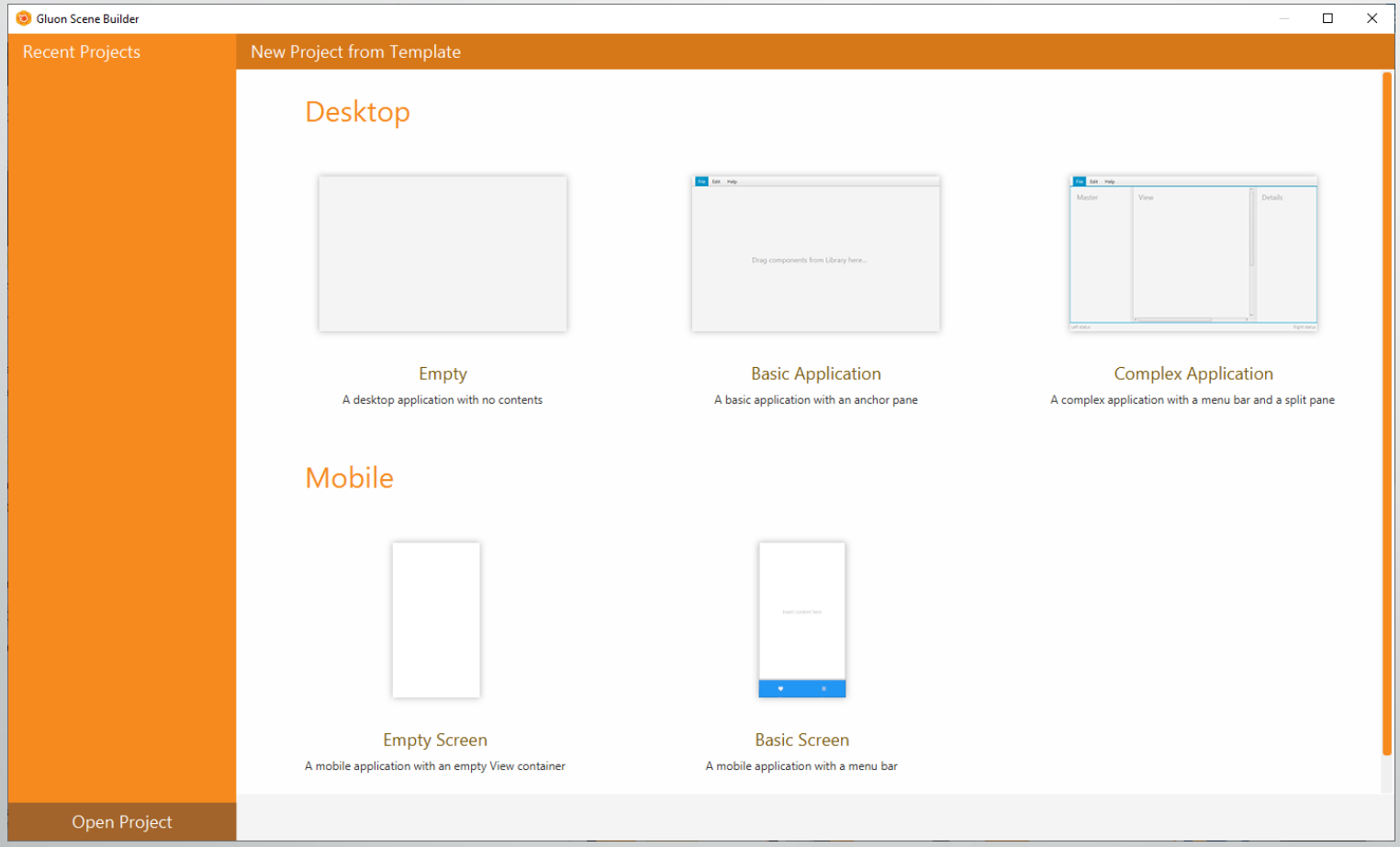
- Pour améliorer l'apparence et le design de votre application, JavaFX permet aussi d'appliquer des effets visuels, tels que :
  - Blend
  - Bloom
  - Blur
  - Drop Shadow
  - Inner Shadow
  - Reflection
  - Lighting
  - Perspective

Et on peut même créer une chaîne d'effets visuels !

Explorer par vous-même (si vous en avez besoin) :

https://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/visual_effects.htm

L'outil Scene Builder

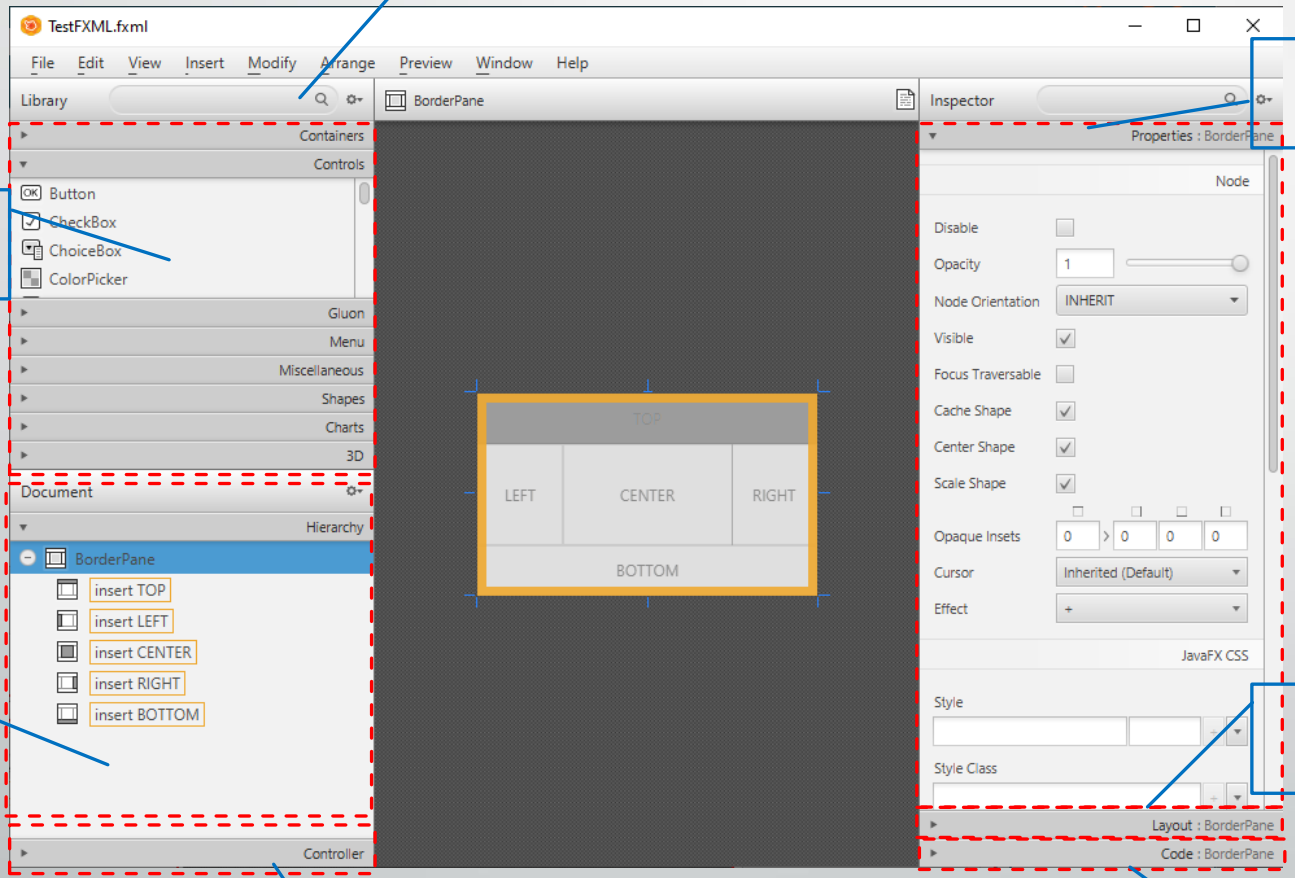


https://gluonhq.com/products/scene-builder/

C’est quoi ? Ça sert à quoi ?

- **Outil graphique officiel :**
  - outil de conception visuelle et interactif
  - gratuit, open source, sous licence BSD
  - distribué par Gluon
- **Alternative à l’approche programmatique**
- **Permet un découplage des responsabilités :**
  - L’interface graphique (vue)
  - La mise en forme (styles css)
  - La logique (classe contrôleur)

Interface de Scene Builder



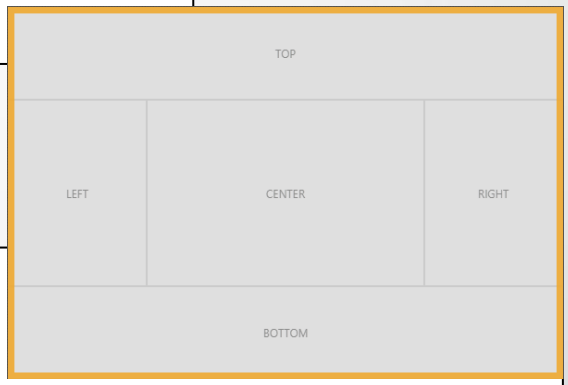
Le fichier FXML

- C'est le support de l'outil Scene Builder : fichier au format XML permettant de décrire votre interface graphique.
- Le nœud racine est généralement un containeur.
- Comme en méthode programmatique : on peut y rattacher des composants ou bien d'autres containeurs (imbrication)
  → le contenu du fichier fxml constitue un graphe de nœuds JavaFX.
- Chaque balise XML est un composant JavaFX, les attributs de cette balise correspondent aux propriétés du composant
- Les noms des balises et des composants coïncident
- Les noms des attributs de la balise et les propriétés du composant coïncident

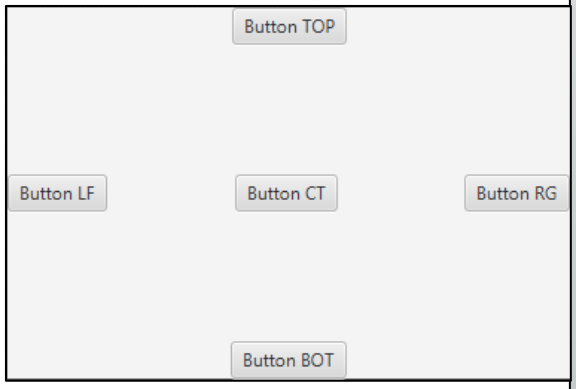
`<Label text="First Name" />` correspond à un composant Label

Exemple d'un fichier FXML

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.BorderPane?>

<BorderPane xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">
    <!-- rattacher des composants JavaFX ici  -->
</BorderPane>
```




```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.BorderPane?>

<BorderPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">
   <center>
      <Button text="Button CT" BorderPane.alignment="CENTER" />
   </center>
   <top>
      <Button text="Button TOP" BorderPane.alignment="CENTER" />
   </top>
   <left>
      <Button text="Button LF" BorderPane.alignment="CENTER" />
   </left>
   <right>
      <Button text="Button RG" BorderPane.alignment="CENTER" />
   </right>
   <bottom>
      <Button text="Button BOT" BorderPane.alignment="CENTER" />
   </bottom>
</BorderPane>
```



Utiliser un fichier FXML

- On va s'appuyer sur la classe utilitaire **FXMLLoader**.
- L'application peut lire et charger ce fichier "**.fxml**" , puis créer le graphe de nœuds qui y est décrit (mécanisme de **désérialisation**).
- On peut ensuite, selon ce que l'on souhaite faire :
  - assigner ce graphe à la racine de notre interface ;
  - ou bien le rattacher à l'un de ses sous-nœuds.

- Le fichier FXML **ne contient pas la logique** de l'application ! Celle-ci est déportée dans une classe à part, qui jouera le rôle de **Contrôleur**.
- L'outil Scene Builder permet d'associer cette classe « contrôleur » avec votre vue.

FXMLLoader

- **Charger un fichier FXML**

```
FXMLLoader.load(getClass().getResource("vue.fxml"));
```
(appel statique)

(ou bien)

```
FXMLLoader loader = new FXMLLoader( );
loader.setLocation(getClass().getResource("vue.fxml"));
loader.load( );
```

Il faudra indiquer correctement le fichier FXML afin qu'il puisse être trouvé et chargé (c'est le même principe que pour le chargement de n'importe quelle ressource).

- **Récupérer le contrôleur** associé à une vue FXML

```
MyViewController ctrl = loader.getController();
```

- **Associer un contrôleur** à une vue FXML

```
loader.setController(new MyViewController());
```

(pas besoin de le déclarer dans l'outil Scene Builder dans ce cas)

https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/FXMLLoader.html

Le Contrôleur

- Classe dans laquelle est déportée la **logique applicative**.
- On peut associer cette classe contrôleur à notre vue FXML :
  - **soit dans Scene Builder** : au niveau du nœud racine (à déclarer dans le volet "*Controller*" situé en bas à gauche) ;
  - **soit en programmatique** : via des méthodes de FXMLLoader.
- C'est une classe Java qui peut contenir :
  - **une méthode d'initialisation** : `private void initialize()`

    (appelée automatiquement après le chargement du fichier FXML)
  - **des attributs liés à la vue fxml** (via les attributs "***fx:id nomAttibut***")

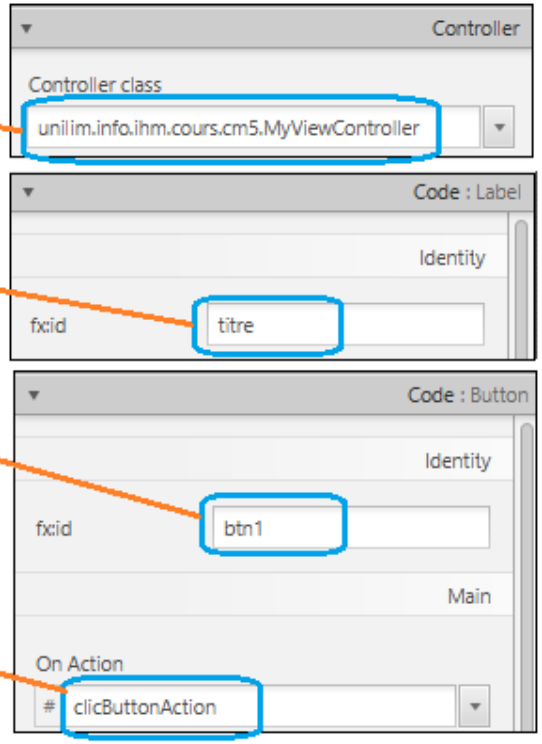
    (dans le volet "*Code*" situé en bas à droite)
  - **des méthodes callback** pour répondre aux évènements

    (dans le volet "*Code*" situé en bas à droite, section "*OnEventXXX*" , par exemple "*OnAction*")
  - **on peut rattacher une feuille de styles CSS** au nœud racine

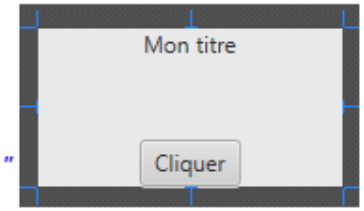
    (dans le volet "*Properties*" situé en haut à droite, section "*JavaFX CSS*")
- Important : chacun de ces éléments devra être **annoté avec `@FML`** (permet le binding entre notre classe contrôleur et le fichier FXML).

Bindings avec Scene Builder

```java
package unilim.info.ihm.cours.cm5;
public class MyViewController {

    // composant injecté par FXMLLoader (fx:id="titre")
    @FXML
    private Label titre;

    // composant injecté par FXMLLoader (fx:id="btn1")
    @FXML
    private Button btn1;

    @FXML
    private void initialize() {
        // initialisation du contrôleur
    }

    @FXML
    void clicButtonAction(ActionEvent event) {
        // callback appelé suite au clic sur le bouton btn1
        System.out.println("Bouton a été cliqué !!!");
        titre.setText("Bonjour !!!");
    }

}
```




```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.BorderPane?>

<BorderPane xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="unilim.info.ihm.cours.cm5.MyViewController">
   <bottom>
      <Button fx:id="btn1" mnemonicParsing="false" onAction="#clicButtonAction" text="Cliquer" />
   </bottom>
   <top>
      <Label fx:id="titre" text="Mon titre" />
   </top>
</BorderPane>
```



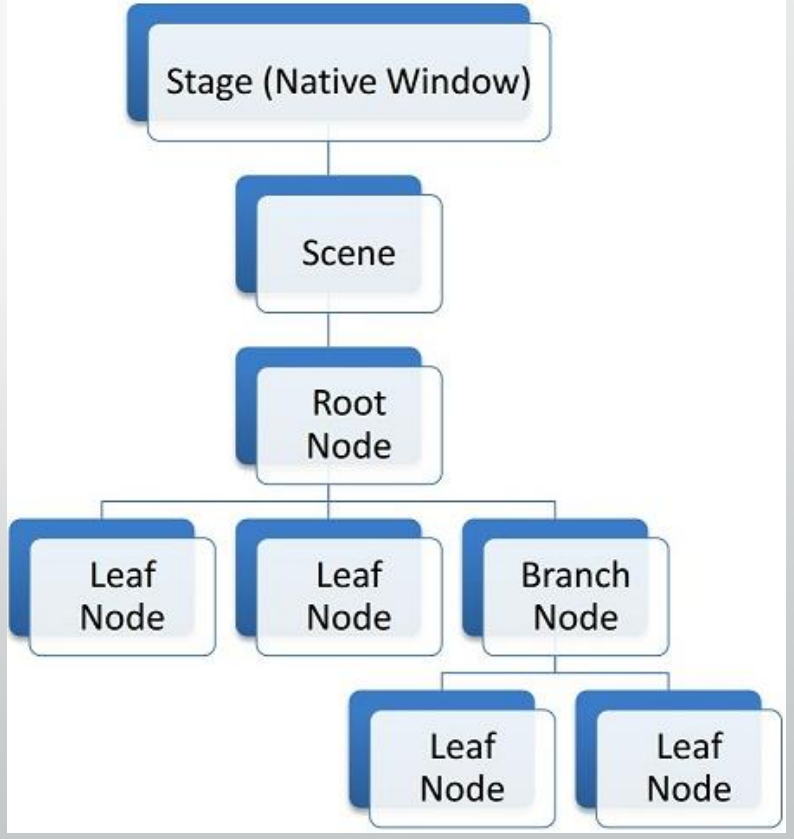
Démonstration de Scene Builder

Gestion de plusieurs fenêtres

- Notre IHM peut avoir besoin de gérer différentes vues
- Rappel : le graphe de scène

Trois niveaux :

- **Stage** : la fenêtre
- **Scene** : une page de l'IHM
- **Nœuds** : racine et enfants



Gestion multi-fenêtres

- Pour changer la vue, plusieurs possibilités :

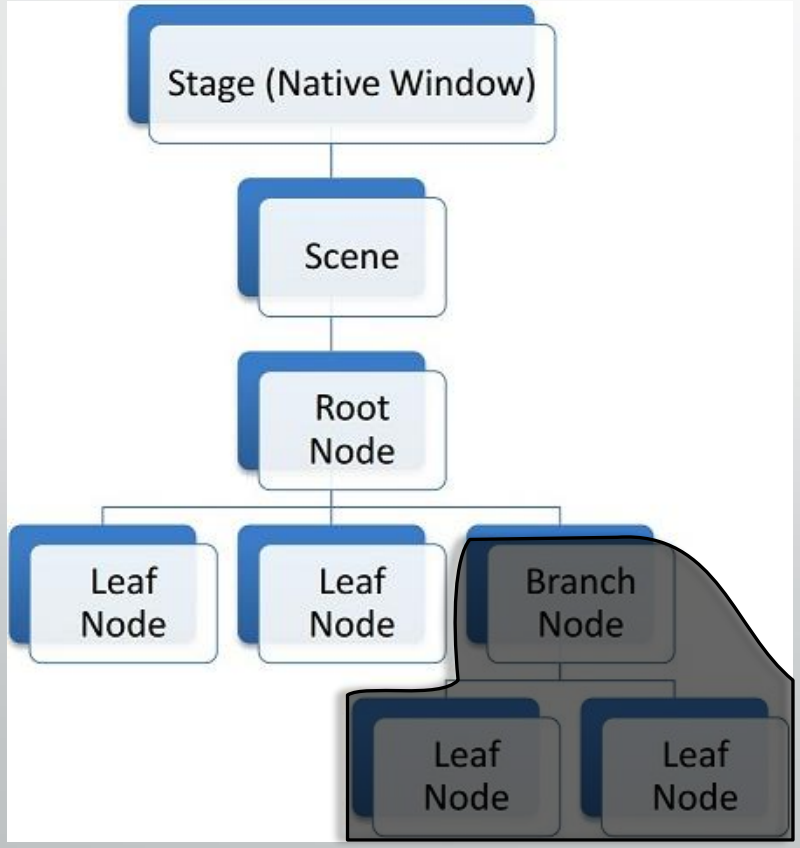
1. Ré-utiliser la même fenêtre et la même scène : remplacer un nœud containeur (Layout Pane) par un autre containeur.

2. Ré-utiliser la même fenêtre : remplacer la scène.

3. Créer une nouvelle fenêtre manuellement.

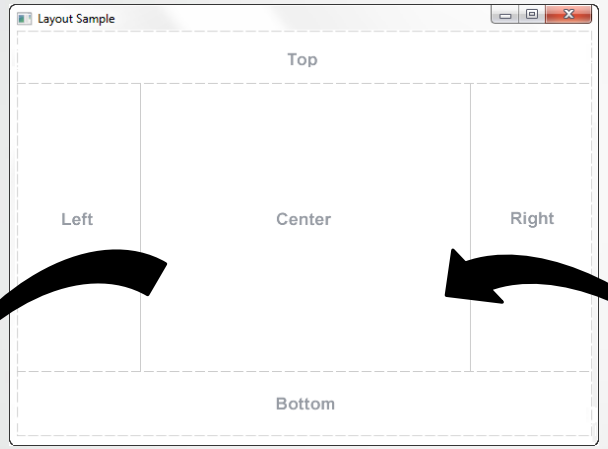
1 - interchanger un nœud

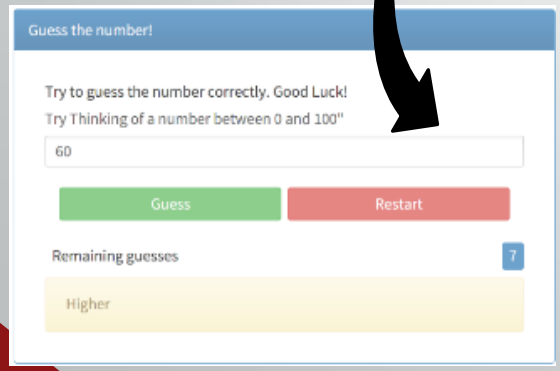
- On peut remplacer un container par un autre.
  - fonctionne pour un container imbriqué
  - ne fonctionne pas pour le nœud racine.

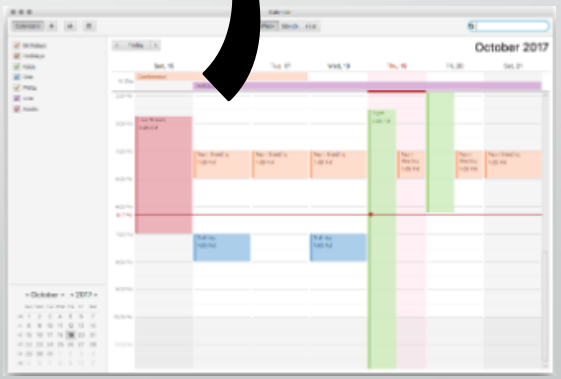


Exemple avec le BorderPane

- On remplace la partie centrale :







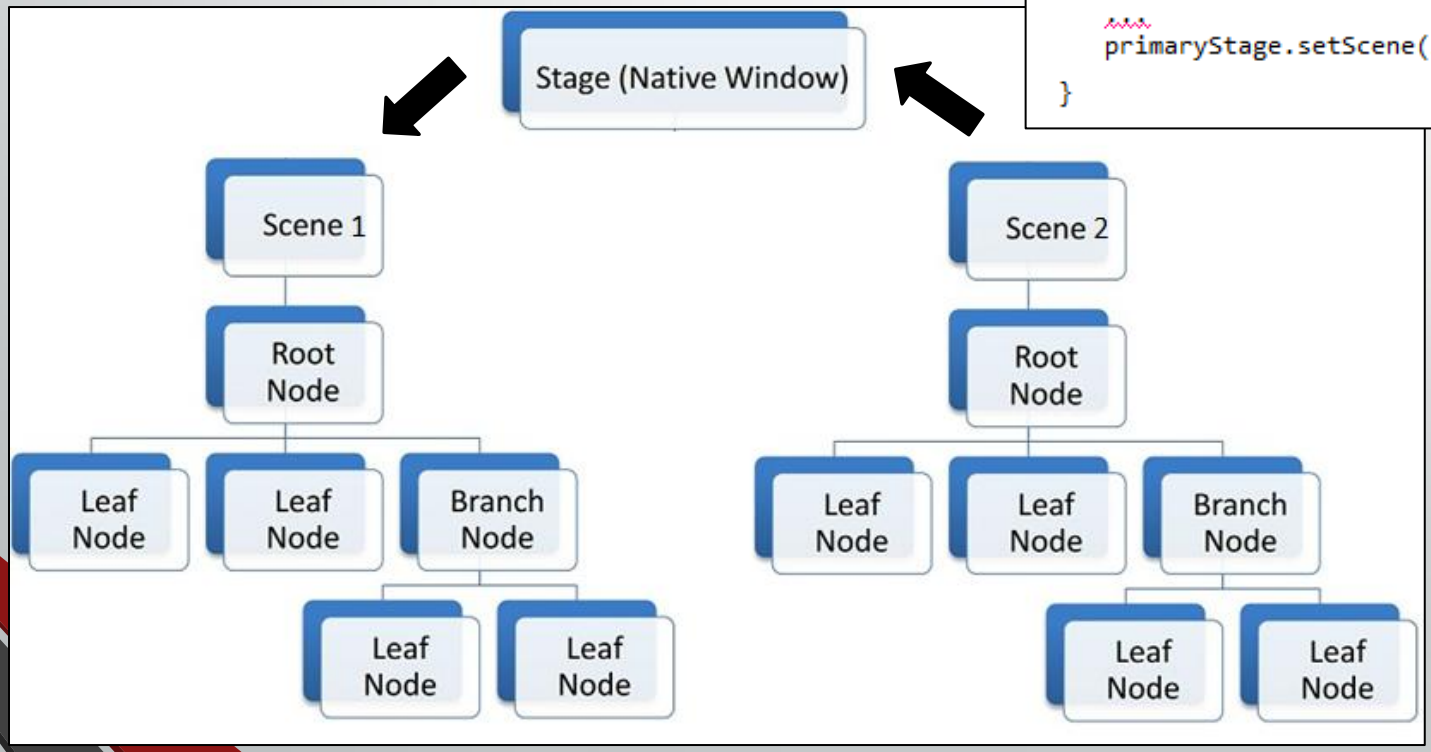
https://github.com/kordamp/bootstrapfx

https://github.com/dlsc-software-consulting-gmbh/CalendarFX

2 - interchanger la Scene

- Dans une même fenêtre, on peut remplacer une Scene par une autre Scene.

```
@Override
public void start(Stage primaryStage)  {

    ...
    scene1 = new Scene(root1, 120, 50);
    ...
    scene2 = new Scene(root2, 240, 25);
    ...
    primaryStage.setScene(scene1);
}
```



3 – créer une toute nouvelle fenêtre

- Créer une nouvelle fenêtre avec la classe **Stage**

```
Stage secondaryStage = new Stage( );
```

- On peut lui associer une fenêtre « parent » (optionnel)

```
secondaryStage.initOwner(primaryStage);
```

- On peut lui définir un style (avant de l'afficher)

```
secondaryStage.initStyle(StageStyle.UNDECORATED);
```

- On peut définir si elle est modale ou pas

```
secondaryStage.initModality(Modality.NONE);
```

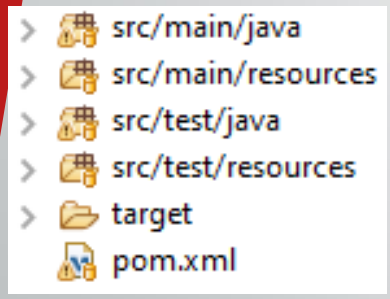
- Enfin il faut lui affecter une scène et l'afficher :

```
secondaryStage.setScene(scene2);  secondaryStage.show( );
```

https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html

Apache Maven

- Si vous voulez utiliser Maven, il faut l'installer si nécessaire :
  https://maven.apache.org/download.cgi
- Outil de construction de projets : **POM** (*Project Object Model*)
- Respecter les dossiers imposés par Maven, par exemple :
  - src/main/java : fichiers sources Java
  - src/main/resources : fichiers ressources (par exemples images)
  - src/test/java : fichiers de test Java
  - src/test/resources : fichiers ressources pour les tests
  - target : répertoire destination pour la compilation
  - fichier pom.xml situé à la racine de votre projet
  - etc



- Si besoin : paramétrer l'intégration de Maven dans votre IDE.
  Sous Eclipse : *Window* > *Preferences* puis *Maven* > *Installations*

Configurer Maven pour JavaFX

- On va se baser sur le plugin d'OpenJFX. Dans le fichier **pom.xml** :
  - il faut déclarer le plugin pour pouvoir l'utiliser :

```
<plugins>
    <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.8</version>
        <configuration>
            <mainClass>org.openjfx.App</mainClass>
        </configuration>
    </plugin>
</plugins>
```

Une autre alternative serait le plugin de Gluon, gluonfx-maven-plugin.

  - Il faut aussi déclarer les dépendances aux modules de JavaFX :

```
<dependencies>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>21.0.1</version>
    </dependency>
    <!--
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>21.0.1</version>
    </dependency>
    -->
</dependencies>
```

A déclarer uniquement si vous manipulez des fichiers FXML

Pas besoin de déclarer la dépendance vis-à-vis du module javafx-graphics (elle sera résolue transitivement par le module javafx-controls)

En cas de problème : il faut vérifier que les plugins et dépendances déclarés sont bien compatibles avec votre version de Maven.

https://openjfx.io/openjfx-docs/#maven

Exemple d'un fichier pom.xml complet

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.openjfx</groupId>
    <artifactId>sample</artifactId>
    <version>1.0.0</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>21.0.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <release>11</release>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
                <configuration>
                    <mainClass>org.openjfx.App</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

GAV = GroupId, ArtifactId et Version à personnaliser pour votre application

Dépendances aux modules JavaFX

Plugin pour compiler vos classes Java

Plugin pour lancer votre application JavaFX, classe main à préciser

JavaFX avec Apache Maven

- Pour lancer votre application, taper la commande suivante :

  mvn clean javafx:run

A noter : la première fois, Maven va télécharger tous les plugins et toutes les dépendances nécessaires.

```
C:\usine-dev\workdir\but-worspace\sample>mvn javafx:run
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< org.openjfx:sample >-------------------------
[INFO] Building sample 1.0.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> javafx-maven-plugin:0.0.8:run (default-cli) > process-classes @ sample >>>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ sample ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\usine-dev\workdir\but-worspace\sample\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ sample ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< javafx-maven-plugin:0.0.8:run (default-cli) < process-classes @ sample <<<
[INFO]
[INFO]
[INFO] --- javafx-maven-plugin:0.0.8:run (default-cli) @ sample ---
[WARNING] Module name not found in <mainClass>. Module name will be assumed from module-info.java
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.594 s
[INFO] Finished at: 2023-10-27T00:35:47+02:00
[INFO] ------------------------------------------------------------------------

C:\usine-dev\workdir\but-worspace\sample>
```

Autres fonctionnalités

- Affichage de contenu HTML 5 (SVG)
- Communication Java – JavaScript : composants WebEngine et WebView
- Multimédia : composants MediaPlayer et MediaView
- Multitouch pour les écrans tactiles
- API d'impression
- Services de concurrence (Task JavaFX)
- Self-Contained Application Packaging
- Interopérabilité avec AWT/Swing grâce au nœud SwingNode

- Et bien d'autres possibilités !

Références

- **Liens utiles**

https://fr.wikipedia.org/wiki/JavaFX

https://openjfx.io/

https://docs.oracle.com/javafx/2/

https://docs.oracle.com/javase/8/javafx/api/

https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

https://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.html

http://gluonhq.com/products/scene-builder/

https://openclassrooms.com/courses/les-applications-web-avec-javafx

http://code.makery.ch/library/javafx-8-tutorial/fr/

https://java.developpez.com/faq/javafx/

- **Livres**

JavaFX for Dummies

Pro JavaFX 8 : A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients (Apress)

Introducing JavaFX 8 Programming (Oracle Press)

Learn JavaFX 8: Building User Experience and Interfaces with Java 8 (Apress)

JavaFX 8: Introduction by Example (Apress)