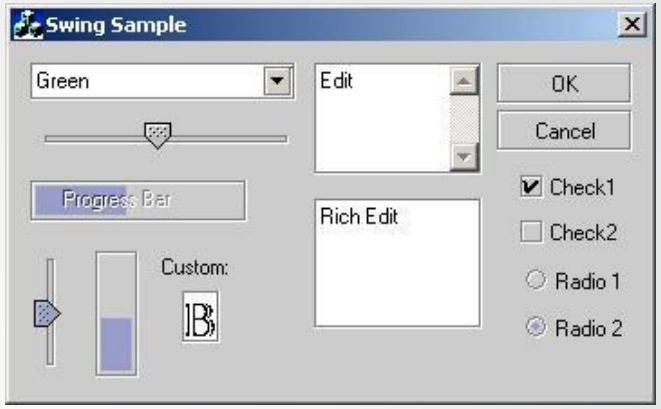
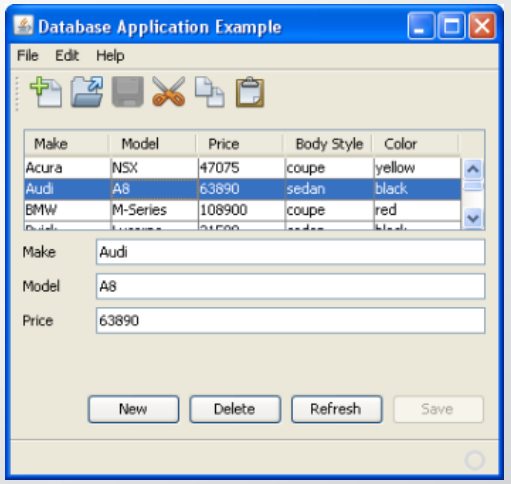
Interfaces homme-machine avec JavaFX

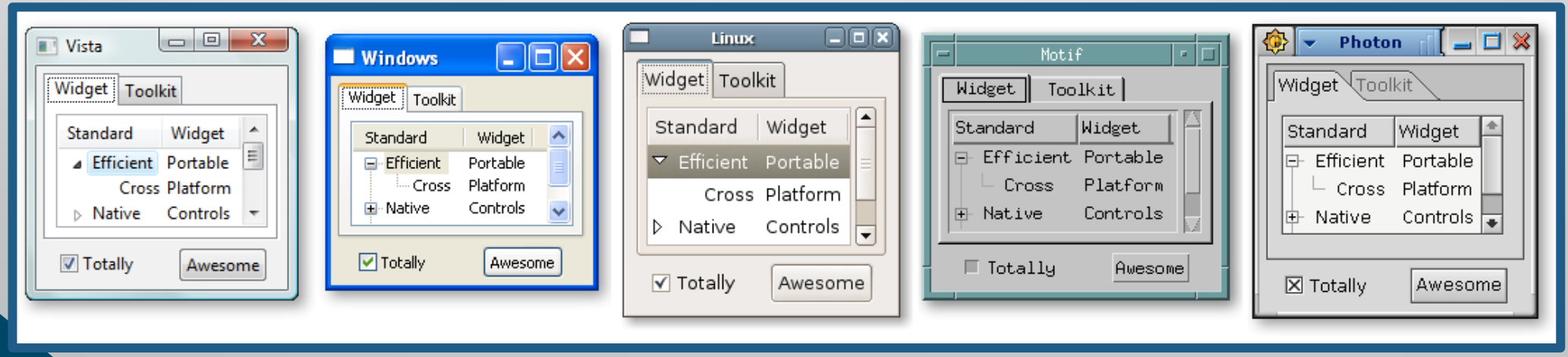
Cette section est un bonus afin de vous présenter une technologie Java pour faire des interfaces graphiques (même si cela n'est pas au cœur de Java et il y a d'autres langages plus appropriés pour faire des IHMs)

JavaFX : Motivational Questions



1. Why does it take a long time to write GUI programs?
2. How can we avoid the "Ugly Java GUI" stereotype **AWT (Abstract Window Toolkit)** → **Swing** → **SWT (Standard Widget Toolkit)**?
3. Why does it seem easier to write **web-apps** than **Swing** programs?
4. And how can I avoid having an enormous, writhing mass of listener patterns?





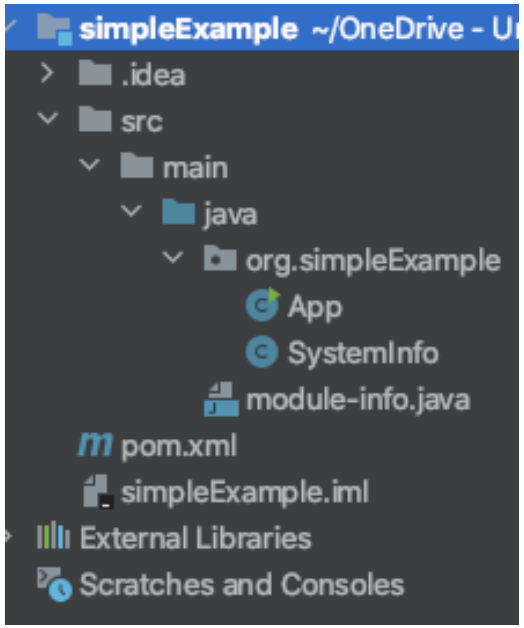
A propos de JavaFX

- JavaFX is **no longer a part of the JDK** starting from JDK 11. That is why, if you use Java 11 and later, you need to download the open-source JavaFX SDK in addition to the JDK from https://gluonhq.com/products/javafx/
- Framework dédié à la conception d'IHM riches (desktop, Internet…)
- Plus moderne et portable que Abstract Windows Toolkit (AWT) et Swing
- En plus des fonctionnalités "standards"
  - Effets et animations poussés
  - Supporte HTML5
  - Supporte CSS (définition de style graphique)
  - Refond (pour le mieux) et étend la notion d'observable en Java

Why JavaFX: https://devm.io/java/20-javafx-real-world-applications-123653
Official documentation: https://docs.oracle.com/javase/8/javafx/api/index.html

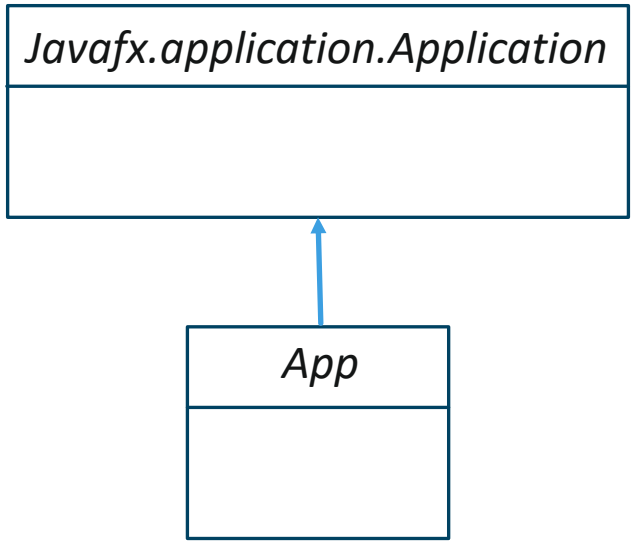
Anatomie d'une application JavaFX (1/4)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        var javaVersion = SystemInfo.javaVersion();
        var javafxVersion = SystemInfo.javafxVersion();
        var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");

        var scene = new Scene(new StackPane(label), 640, 480);
        stage.setScene(scene);

        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```



Anatomie d'une application JavaFX (2/4)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        var javaVersion = SystemInfo.javaVersion();
        var javafxVersion = SystemInfo.javafxVersion();
        var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");

        var scene = new Scene(new StackPane(label), 640, 480);
        stage.setScene(scene);

        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```



Anatomie d'une application JavaFX (3/4)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        var javaVersion = SystemInfo.javaVersion();
        var javafxVersion = SystemInfo.javafxVersion();
        var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");

        var scene = new Scene(new StackPane(label), 640, 480);
        stage.setScene(scene);

        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

Si vous avez un constructeur, attention à ne pas initialiser les éléments graphiques là (dans le constructeur).

Attention à ne pas initialiser les éléments graphiques dans le *main* car au moment de l'appel à la méthode main JavaFX n'est pas encore chargé.
Le *lunch()* est optionnel ou non selon les IDEs.

Anatomie d'une application JavaFX (4/4)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        var javaVersion = SystemInfo.javaVersion();
        var javafxVersion = SystemInfo.javafxVersion();
        var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");

        var scene = new Scene(new StackPane(label), 640, 480);
        stage.setScene(scene);

        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

1) Initialisation des éléments graphiques

2) Définition du « layout »

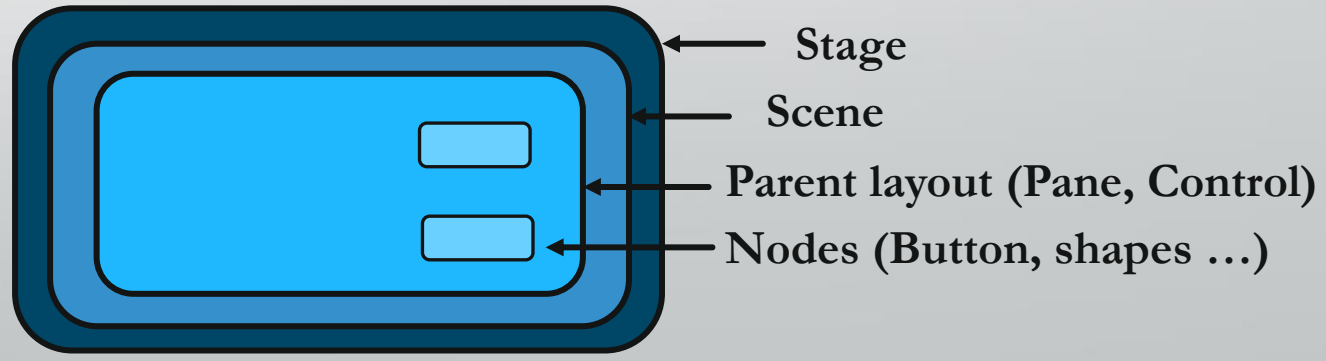
3) Lancement de l'application

Lancement de l'application

Différences entre stage et scène

- Le *stage* d'une application ne change pas lors de son exécution.
- Le *stage* ne contient qu'une seule scène, laquelle peut changer.
- **Le "Stage" est le conteneur** de l'application, **la "Scene" en est le mode actuel** (ex: chargement, fonctionnement nominal).

Exemple: L'écran de chargement de votre IDE et son éditeur une fois lancé sont deux scènes contenues successivement par le même *stage*.



Lancement de l'application

La séquence d'actions à effectuer

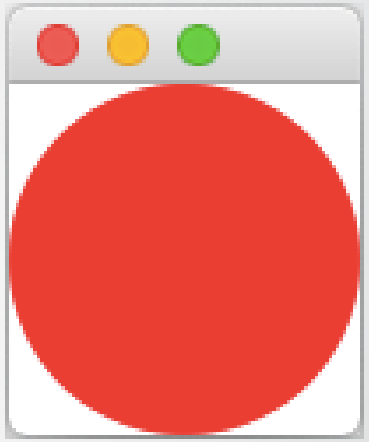
```
Scene myScene = new Scene (myPane, 300, 50);
stage.setScene(myScene);
stage.setTitle("Hello world application");
stage.show();
```

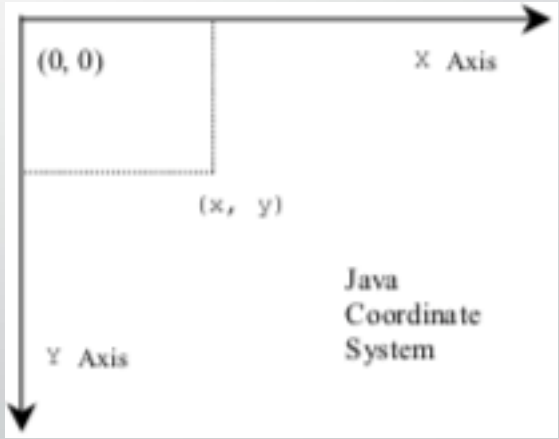
1. **Création de la scène** avec son « layout » (le *Pane* principal) et sa taille.
2. **Paramétrage du stage** (sa scène, son titre et d'autres options comme interdire d'en changer la taille)
3. ***stage.show();***

JavaFX 2.x

FXCircle.java

```
import javafx.application.Application;
import javafx.scene.*;

public class FXCircle extends Application {

@Override public void start(Stage stage) {
   Group root = new Group();
   Scene scene = new Scene(root,100,100);
   stage.setScene(scene);
   Circle circle = new Circle();
   circle.setRadius(50f);
   circle.setCenterX(50f);
   circle.setCenterY(50f);
   circle.setFill(Color.RED);
   root.getChildren().add(circle);
   stage.show();
 }
 public static void main(String[] args) {
   launch(args);
 }
}
```





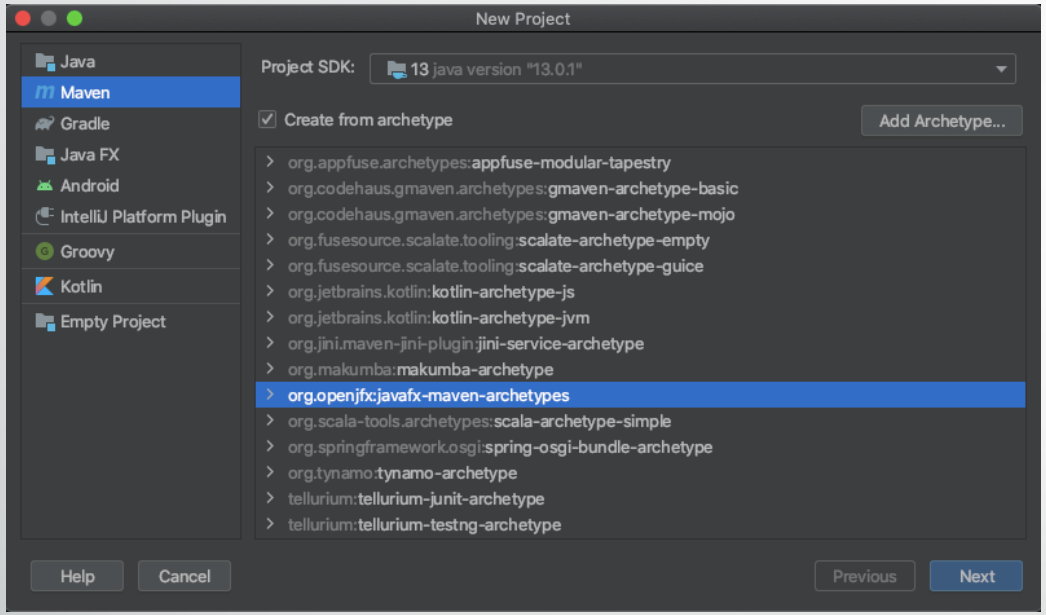
f means pixels

Creating and configuring our first project in IntelliJ with FXML

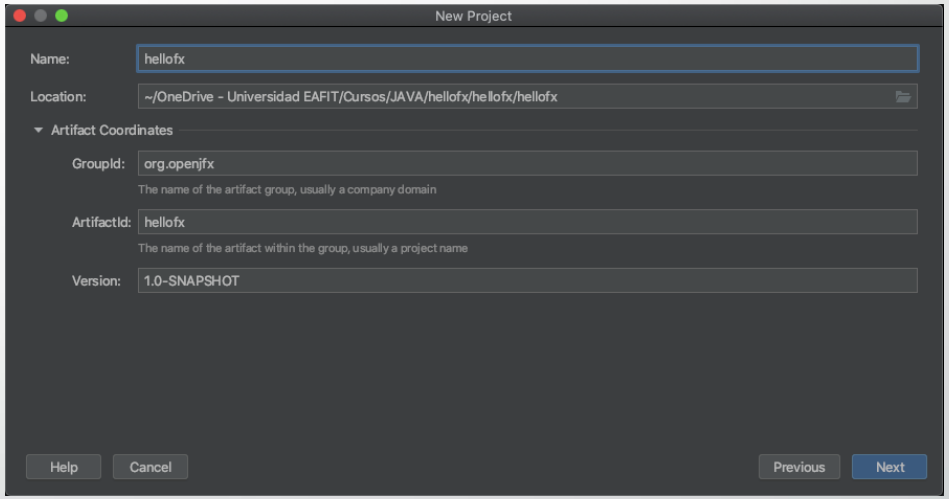
1. Download the last JavaFX SDK suitable for your operating system: https://gluonhq.com/products/javafx/

2. Unpack the archive and place the folder in your file system, for example: **/Users/Desktop/javafx-sdk-15.0.1**

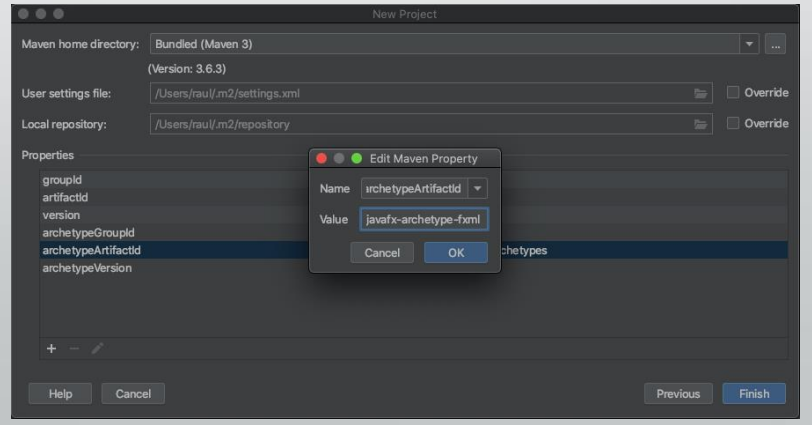
3. Select File -> New -> Project -> Maven and enable Create from archetype. If the JavaFX archetype is not installed yet, select Add archetype... and set the groupId (org.openjfx), the artifactId (javafx-maven-archetypes), and the version (0.0.5), press OK and select the artifact.



4. Provide the groupId, like org.openjfx, the artifactId, like hellofx.



5. Select javafx-archetype-fxml as the value of the archetypeArtifactId, then OK, then click on Finish.



6. Verify that the pom.xml includes the javafx.controls and javafx.fxml dependencies, and includes the javafx-maven-plugin. Open the module-info class, that includes the required modules javafx.controls and javafx.fxml.

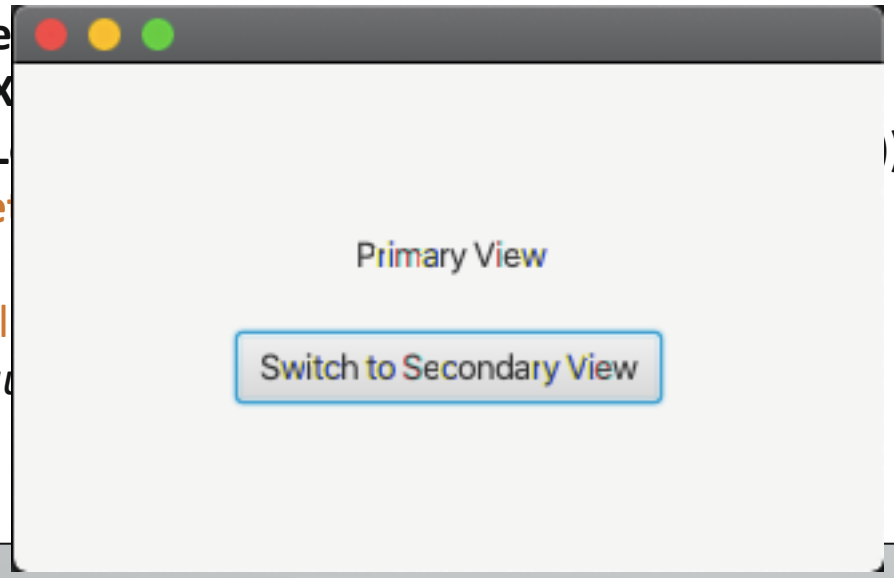
JavaFX with FXML

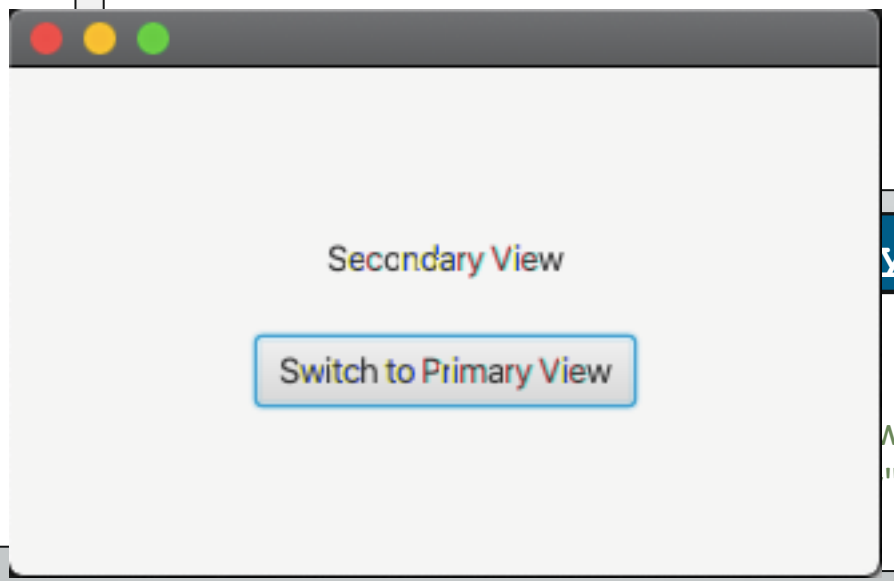
App.java

```
public class App extends Application {
    private static Scene scene;
    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML("primary"), 640, 480);
        stage.setScene(scene);
        stage.show();
    }
    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }
    private static Parent loadFXML(String fxml) throws
IOExce
        FX
FXMLL                                              );
        re
    }
    publ
        la
    }
}
```

primary.xml

```
<VBox alignment="CENTER" spacing="20.0"
xmlns="http://javafx.com/javafx/8.0.171"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.openjfx.PrimaryController">
  <children>
    <Label text="Primary View" />
    <Button fx:id="primaryButton"
text="Switch to Secondary View"
onAction="#switchToSecondary"/>
  </children>
  <padding>
    <Insets bottom="20.0" left="20.0"
```





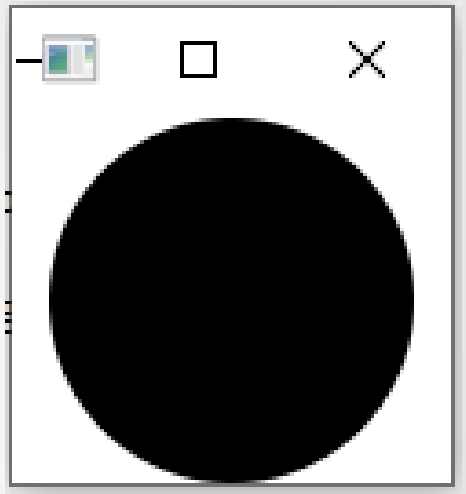
y.xml

```
witch to
" />
```

JavaFX with FXML

JavaFXTest.java

```
import java.net.URL; import
javafx.application.Application;
import javafx.fxml.FXMLLoader; import javafx.scene.*;
import javafx.stage.Stage;

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

  @Override
  public void start(Stage stage) {
    stage.setTitle("FXML Example");
    try{
      Parent root = FXMLLoader.load(new
URL("file:///E:/circle.xml"));
      stage.setScene(new Scene(root));
      stage.show();
    }
   catch(Exception e){
     System.out.println(e.getMessage());
   }
  }
}
```



Circle.xml

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

<?import javafx.scene.control.*?>
<?import javafx.scene.shape.*?>
<?import javafx.scene.layout.BorderPane?>

<BorderPane>
<center>
<Circle radius="50" centerX="50"
centerY="50" />
</center>
</BorderPane>
```

Java API and FXML

Java API for JavaFX

- End-to-end Java development
- Leverage Java language features – generics, annotations, threading, etc.
- Fluent style API for UI construction
- Alternative JVM languages support (eg Groovy, Scala) with JavaFX
- Leverage sophisticated IDEs, debuggers and profilers

FXML

- Scriptable, XML based language for defining UI
- Convenient alternative to developing UI programmatically
- Easy to learn and intuitive for developers familiar with web technologies or other markup based UI technologies
- Allow any JVM based scripting language to be embedded within FXML

Eléments graphiques

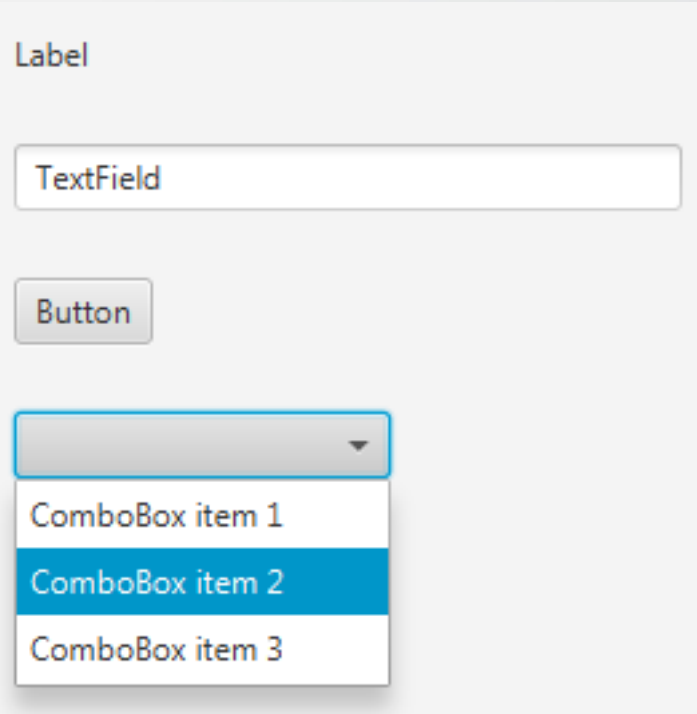
1. Définir les éléments. Quelques exemples :
   - Les simples et classiques

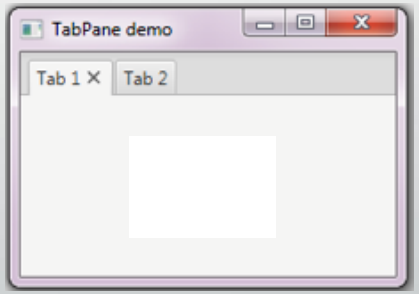
     Label

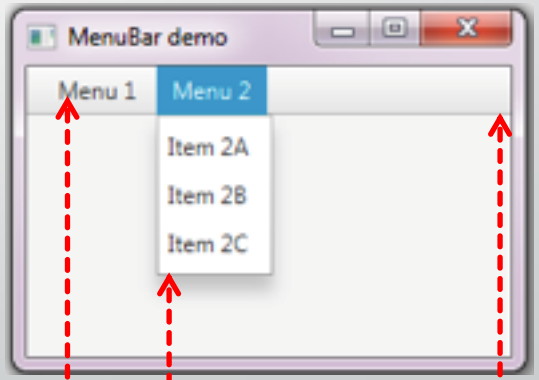
     TextField

     Button

     ComboBox
   - *MenuBar, Menu, MenuItem*
   - *TabPane*
   - *Shapes*







2. Définir le comportement de chaque élément
   - MenuItem
   - MenuBar
   - Menu

Eléments graphiques
Les simples et classiques

Label

```
Label myLabel = new Label ("Hello world!");
```

TextField

```
TextField myTextField = new TextField("By default TextField text");
```

Button

```
Button myButton = new Button ("Button label");
```

ComboBox

```
ComboBox<String> myComboBox = new ComboBox<>();
myComboBox.getItems().addAll("ComboBox item 1", "ComboBox item 2", "ComboBox item 3");
```

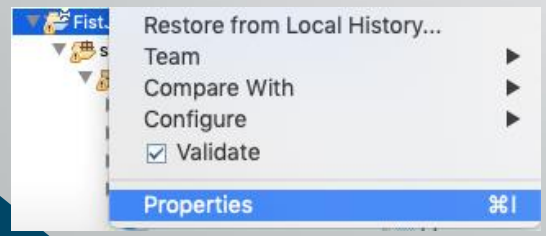
Eléments graphiques stylisés avec CSS

```
public void start(Stage stage) {
    Text emailText = new Text("Email");
    Text passwordText = new Text("Password");
    TextField emailTF = new TextField();
    PasswordField passwordTF = new PasswordField();
    Button submitB = new Button("Submit");
    Button clearB = new Button("Clear");
    GridPane gridPane = new GridPane();
    gridPane.setMinSize(400, 200);

    //Setting the padding between columns of the GridPane
    gridPane.setPadding(new Insets(10, 10, 10, 10)); //margins around the whole grid (top/right/bottom/left)
    //Setting the vertical and horizontal gaps between the columns
    gridPane.setVgap(5);
    gridPane.setHgap(5);
    //Setting the Grid alignment
    gridPane.setAlignment(Pos.CENTER);

    //Arranging all the nodes in the grid
    gridPane.add(emailText, 0, 0);//col:0, row:0
    gridPane.add(emailTF, 1, 0);//col:1, row:0
    gridPane.add(passwordText, 0, 1);//col:0, row:1
    gridPane.add(passwordTF, 1, 1);//col:1, row:1
    gridPane.add(submitB, 0, 2);//col:0, row:2
    gridPane.add(clearB, 1, 2);//col:1, row:2

    //Styling nodes
    submitB.setStyle("-fx-background-color: Grey; -fx-text-fill: white;");
    clearB.setStyle("-fx-background-color: Grey; -fx-text-fill: white;");
    emailText.setStyle("-fx-font: normal bold 20px 'serif' ");
    passwordText.setStyle("-fx-font: normal bold 20px 'serif' ");
    gridPane.setStyle("-fx-background-color: ALICEBLUE;");

    Scene scene = new Scene(gridPane);
    stage.setTitle("Exemple CSS sans fichier CSS");
    stage.setScene(scene); stage.show();
}
```

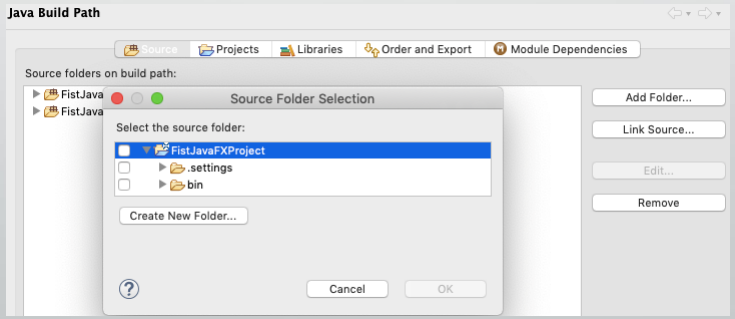
Pour les noms des couleurs:
https://www.w3schools.com/colors/colors_names.asp

Eléments graphiques stylisés avec CSS file

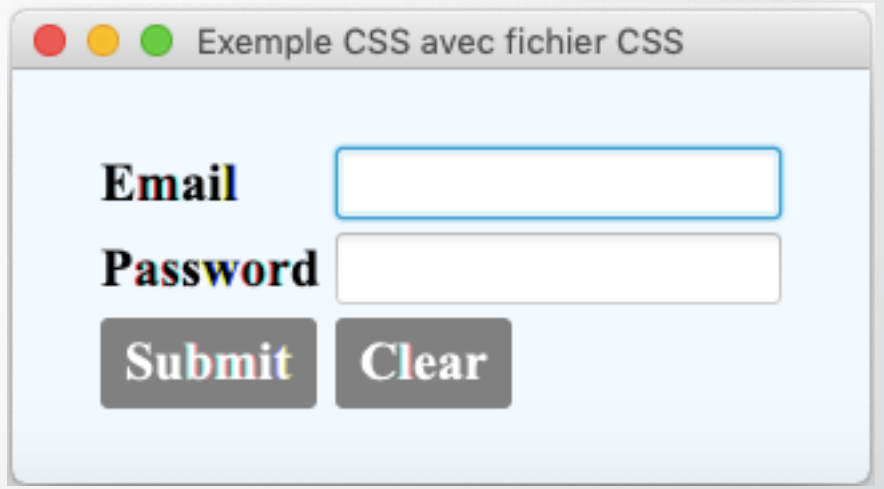
```
public void start(Stage stage) {
    Text emailText = new Text("Email");
    Text passwordText = new Text("Password");
    TextField emailTF = new TextField();
    PasswordField passwordTF = new PasswordField();
    Button submitB = new Button("Submit");
    Button clearB = new Button("Clear");
    GridPane gridPane = new GridPane();
    gridPane.setMinSize(400, 200);
    gridPane.setPadding(new Insets(10, 10, 10, 10));
    gridPane.setVgap(5); gridPane.setHgap(5);
    gridPane.setAlignment(Pos.CENTER);
    gridPane.add(emailText, 0, 0);//col:0, row:0
    gridPane.add(emailTF, 1, 0);//col:1, row:0
    gridPane.add(passwordText, 0, 1);//col:0, row:1
    gridPane.add(passwordTF, 1, 1);//col:1, row:1
    gridPane.add(submitB, 0, 2);//col:0, row:2
    gridPane.add(clearB, 1, 2);//col:1, row:2
    //set the ccs style classes and identifiers
    emailText.getStyleClass().add("text");
    passwordText.getStyleClass().add("text");
    gridPane.setId("grid-pane");
    Scene scene = new Scene(gridPane, 500, 400);
    scene.getStylesheets().add("application.css");
    stage.setTitle("Exemple CSS avec fichier CSS");
    stage.setScene(scene);
    stage.show();
}
```



Right click -> Properties



Java Build Path -> Source (Tab) -> Add Folder -> **Create New Folder, call it: resources**



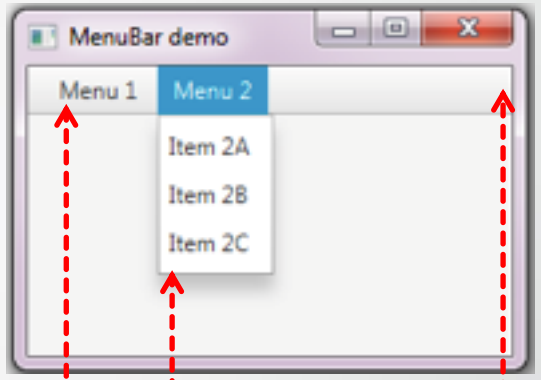
Put this file in the “resources” package:

application.css

```
.button {
  -fx-background-color: grey;
  -fx-text-fill: white;
}
.text {
  -fx-font: normal bold 20px 'serif';
}
#grid-pane {
  -fx-background-color: ALICEBLUE;
}
```

Eléments graphiques

MenuBar, Menu, MenuItem

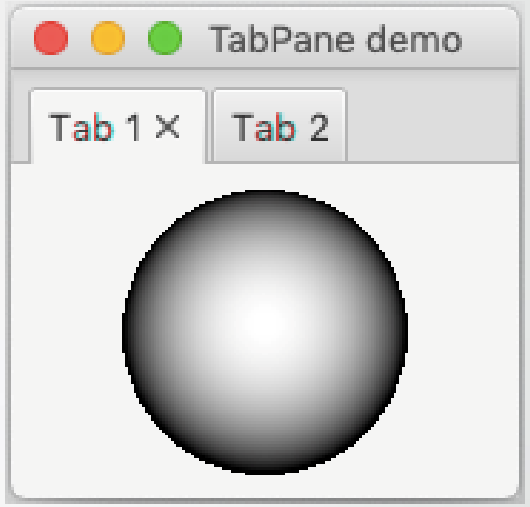


- MenuBar
- Menu
- MenuItem

```
MenuItem myMenuItem1A = new MenuItem("Item 1A");
MenuItem myMenuItem1B = new MenuItem("Item 1B");
MenuItem myMenuItem2A = new MenuItem("Item 2A");
MenuItem myMenuItem2B = new MenuItem("Item 2B");
MenuItem myMenuItem2C = new MenuItem("Item 2C");

Menu myMenu1 = new Menu ("Menu 1");
myMenu1.getItems().addAll(myMenuItem1A, myMenuItem1B);
Menu myMenu2 = new Menu ("Menu 2");
myMenu2.getItems().addAll(myMenuItem2A, myMenuItem2B, myMenuItem2C);

MenuBar myMenuBar = new MenuBar();
myMenuBar.getMenus().addAll(myMenu1, myMenu2);
Pane myPane = new VBox();
myPane.getChildren().add(myMenuBar);
```

Eléments graphiques

TabPane



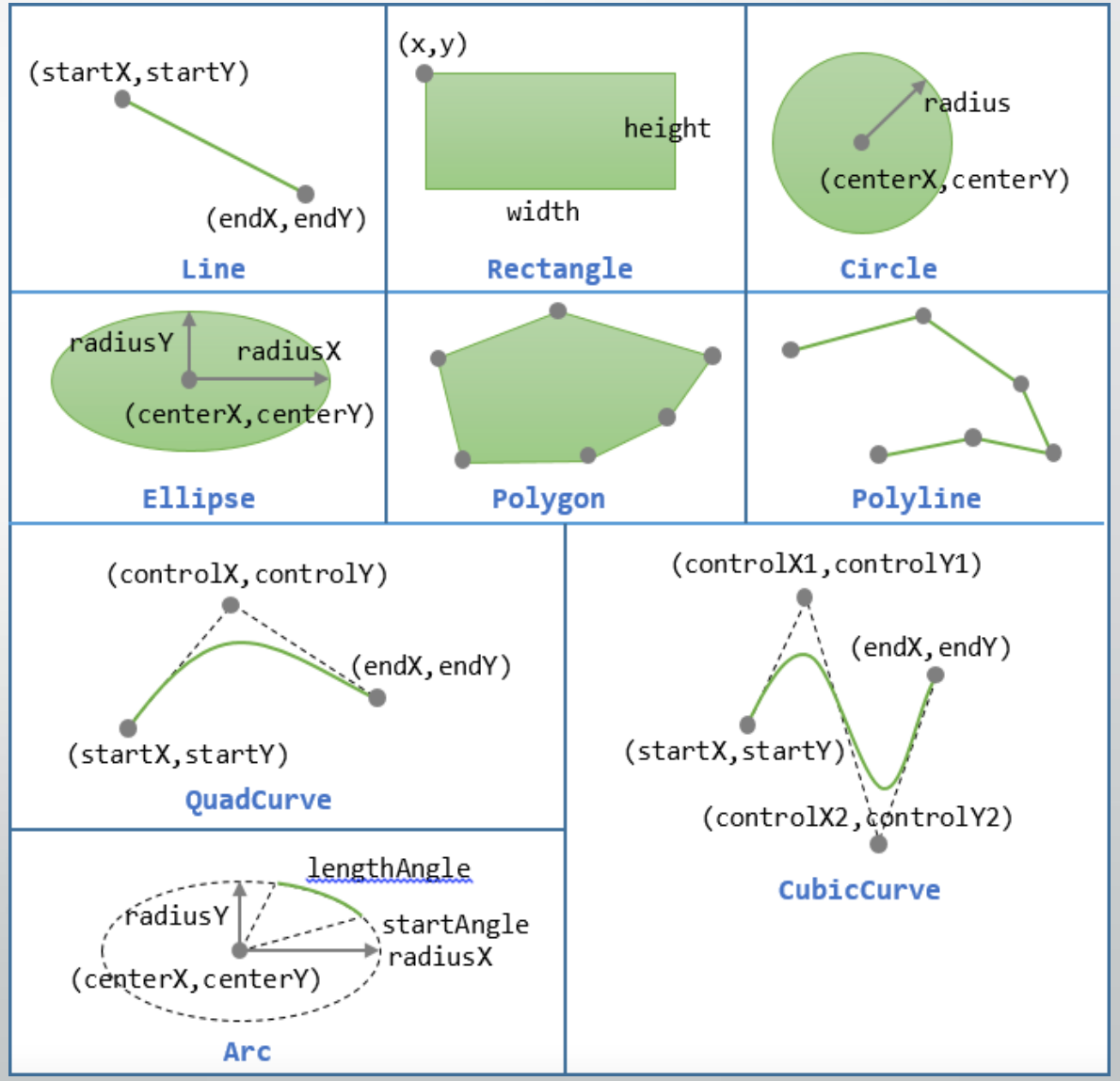

```
public void start(Stage stage) {
        Tab myTab1 = new Tab("Tab 1");
        Pane myTab1Pane = new StackPane();
        Sphere sphere = new Sphere();
        sphere.setRadius(50.0);
        myTab1Pane.getChildren().add(sphere);
        myTab1.setContent(myTab1Pane);

        Tab myTab2 = new Tab("Tab 2");
        Pane myTab2Pane = new StackPane();
        myTab2Pane.getChildren().add(new Circle(25,     Color.DODGERBLUE));
        myTab2.setContent(myTab2Pane);

        TabPane tabPane = new TabPane();
        tabPane.getTabs().addAll(myTab1, myTab2);
        Scene myScene = new Scene (myTabPane, 250, 150);
        stage.setScene(myScene);
        stage.setTitle("TabPane demo");
        stage.show();
}
```

Eléments graphiques

Shapes

Les constructeurs :

Line(**double** startX, **double** startY, **double** endX, **double** endY)
Rectangle(**double** x, **double** y, **double** width, **double** height)
Circle(**double** centerX, **double** centerY, **double** radius)
Ellipse(**double** centerX, **double** centerY, **double** radiusX, **double** radiusY)
Polygon(**double**[] xyPairs)
Polyline(**double**[] xyPairs)
QuadCurve(**double** startX, **double** startY, **double** controlX, **double** controlY, **double** endX, **double** endY)
CubicCurve(**double** startX, **double** startY, **double** controlX1, **double** controlY1, **double** controlX2, **double** controlY2, **double** endX, **double** endY)
Arc(**double** centerX, **double** centerY, **double** radiusX, **double** radiusY, **double** startAngleDegree, **double** lengthAngleDegree)



Eléments graphiques
Définir le comportement (1/8)

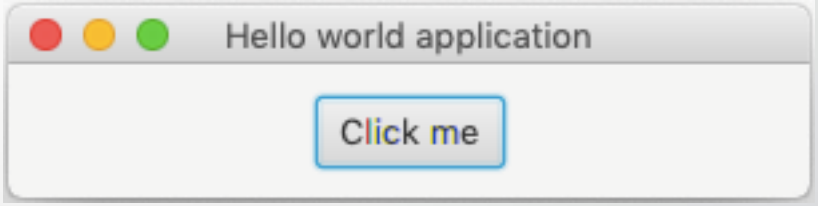
En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)

Eléments graphiques
Définir le comportement (2/8)

En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)

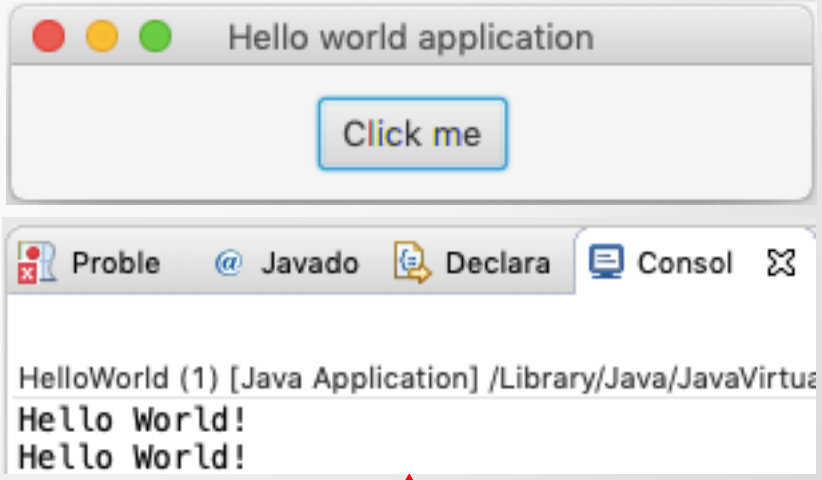



```
Button myButton = new Button("Click me");
//ToDo myButton behavior

Pane myPane = new StackPane();
myPane.getChildren().add(myButton);
Scene myScene = new Scene (myPane, 300, 50);
stage.setScene(myScene);
stage.setTitle("Hello world application");
stage.show();
```

Eléments graphiques
Définir le comportement (3/8)

En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)



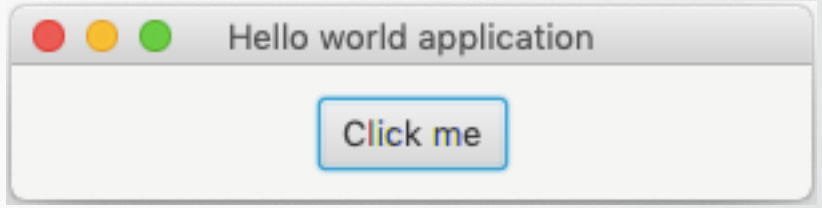

```
Button myButton = new Button("Click me");
//ToDo myButton behavior

Pane myPane = new StackPane();
myPane.getChildren().add(myButton);
Scene myScene = new Scene (myPane, 300, 50);
stage.setScene(myScene);
stage.setTitle("Hello world application");
stage.show();
```

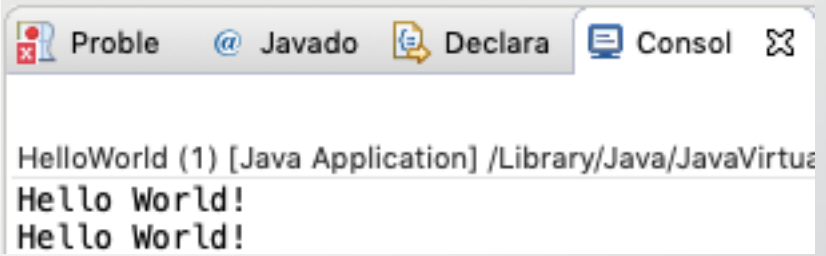
Eléments graphiques

Définir le comportement (4/8)

En gros, trois possibilités:

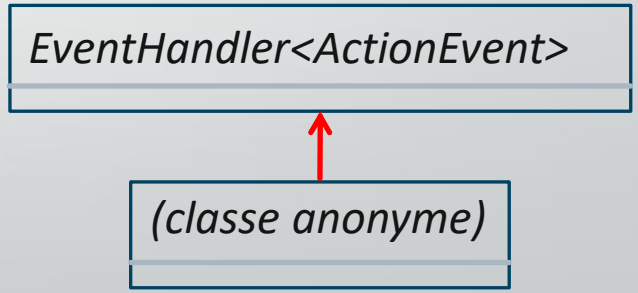
- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)






```
Button myButton = new Button("Click me");
//myButton behavior
myButton.setOnAction(
        new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        }
);
```

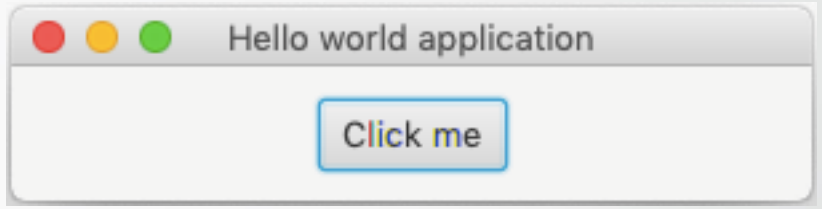
Peu lisible!

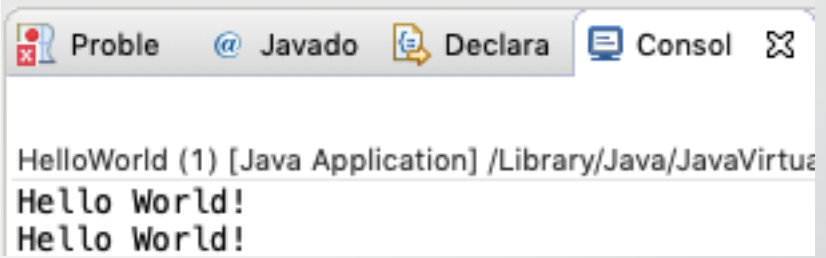


Eléments graphiques
Définir le comportement (5/8)

En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)



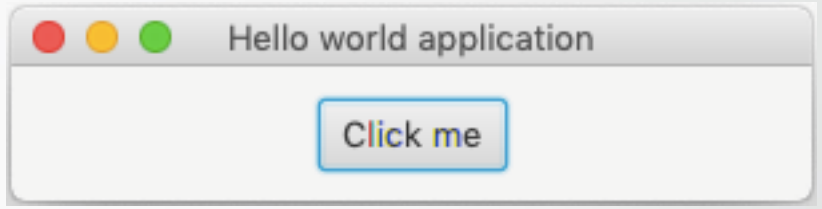


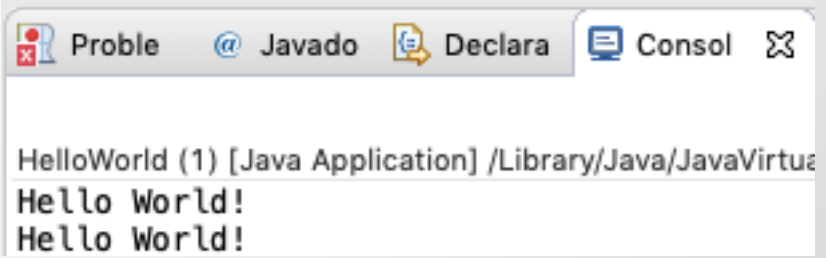

```
Button myButton = new Button("Click me");
//myButton behavior
myButton.setOnAction(actionevent -> {
        System.out.println("Hello World!");
        //ici toute la logique du comportement de myButton
});
```

Eléments graphiques
Définir le comportement (6/8)

En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)






```
Button myButton = new Button("Click me");
//myButton behavior
myButton.setOnAction(this::myButtonBehavior);
```

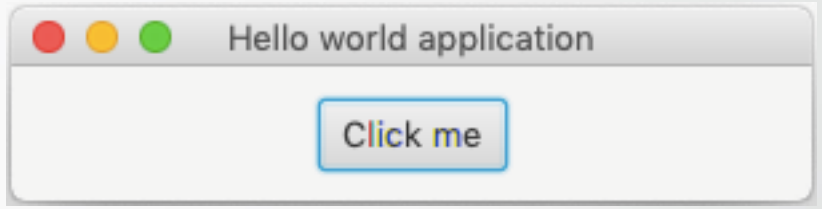
Et à l'extérieur de la méthode `public void start(Stage stage){}` :

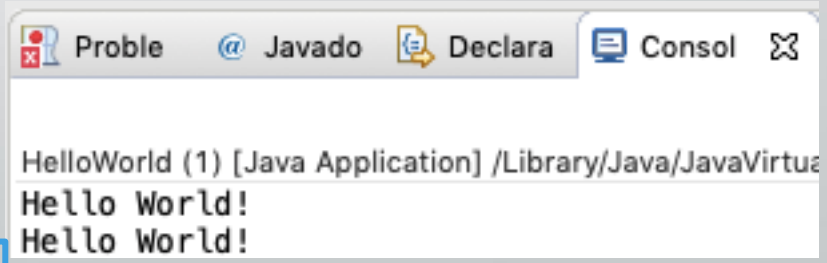
```
private void myButtonBehavior(ActionEvent event) {
    System.out.println("Hello World!");
}
```

Eléments graphiques

Définir le comportement (7/8)

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

        @Override
        public void start(Stage stage) throws Exception {
                try {
                        Button myButton = new Button("Click me");
                        myButton.setOnAction(this::myButtonBehavior);
                        Pane myPane = new StackPane();
                        myPane.getChildren().add(myButton);
                        Scene myScene = new Scene (myPane, 300, 50);
                        stage.setScene(myScene);
                        stage.setTitle("Hello world application");
                        stage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }
        private void myButtonBehavior(ActionEvent event){
                System.out.println("Hello World!");
        }
}
```





Eléments graphiques
Définir le comportement (8/8)

En gros, trois possibilités:

- Classes anonymes (JDK >= 1.7)
- Lambda expressions (JDK >= 1.8)
- Pointeurs de méthode (JDK >= 1.8)

Les pointeurs de méthode est l'option recommandée car:

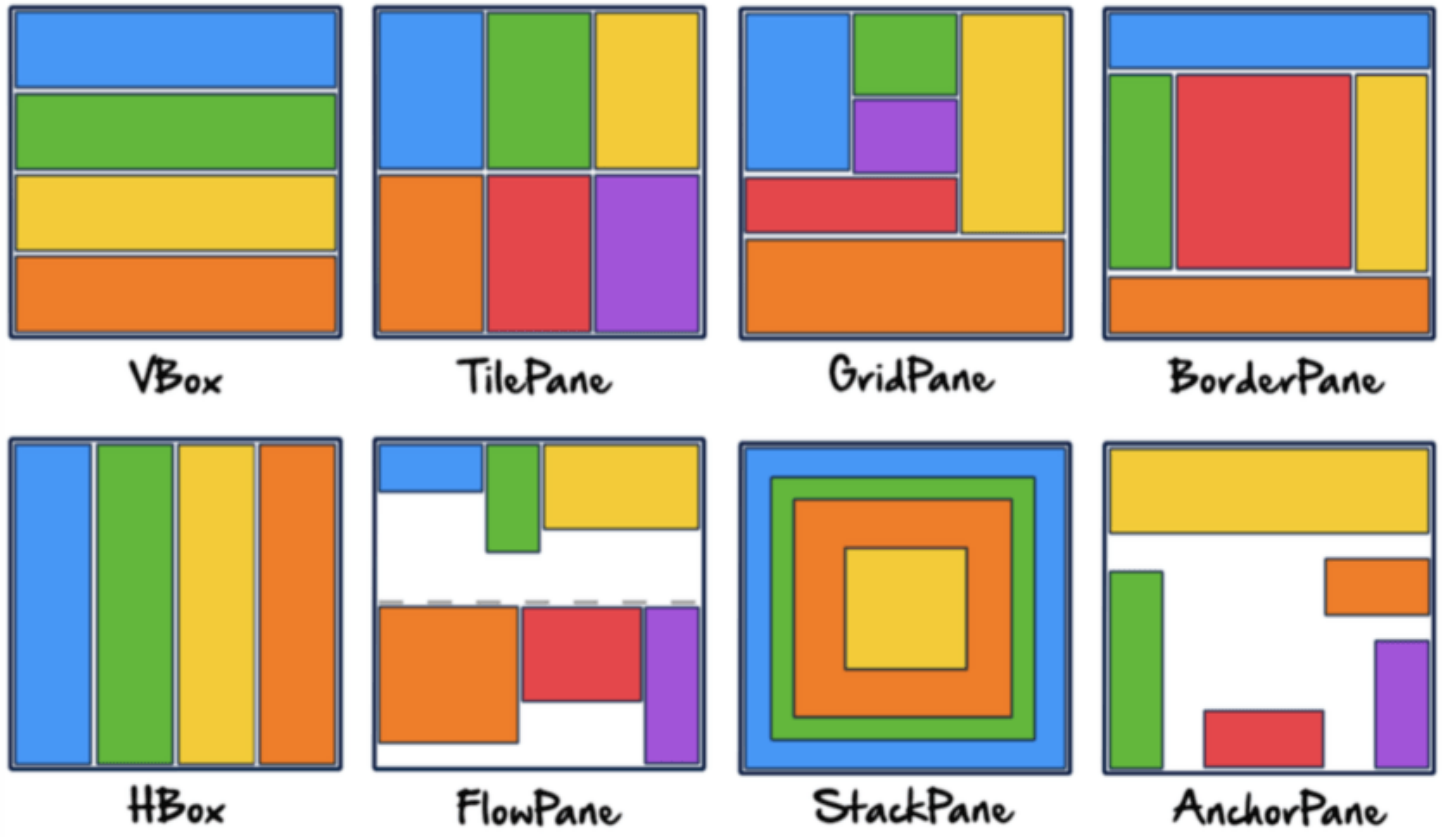
- Force la création de méthodes dédiées (+réutilisable)
- Vous permet de nommer vos comportements (+lisibilité)

« Layout » (1/4)

Réalisé via conteneurs dédiés appelés *Pane* :

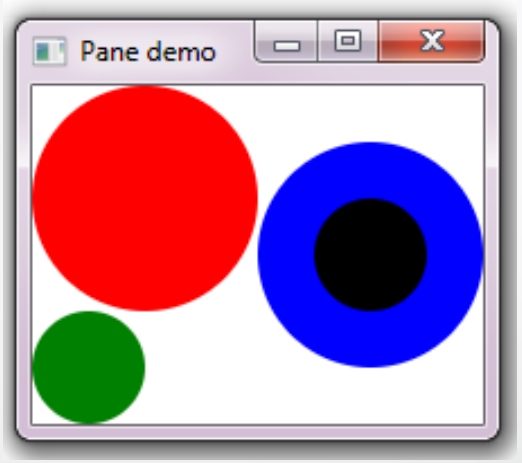
LAYOUT PANES

JavaFX provides several layouts out of the box, which can be seen in the following diagram:



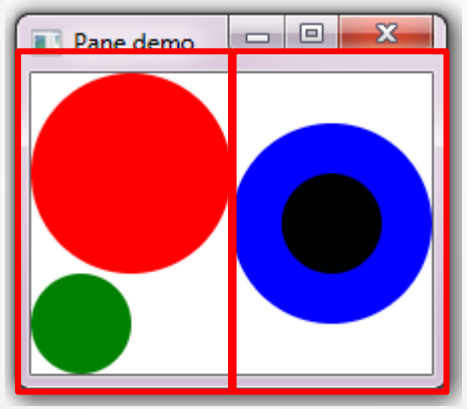
« Layout » (2/4)

Un *Pane* peut en cacher un autre!

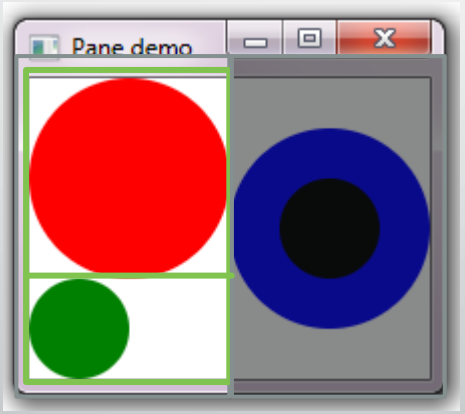


« Layout » (3/4)

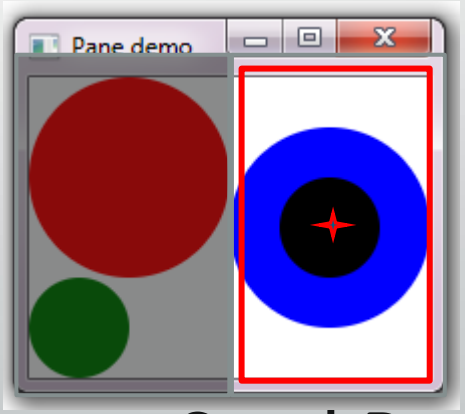
Un *Pane* peut en cacher un autre!



HBox



VBox

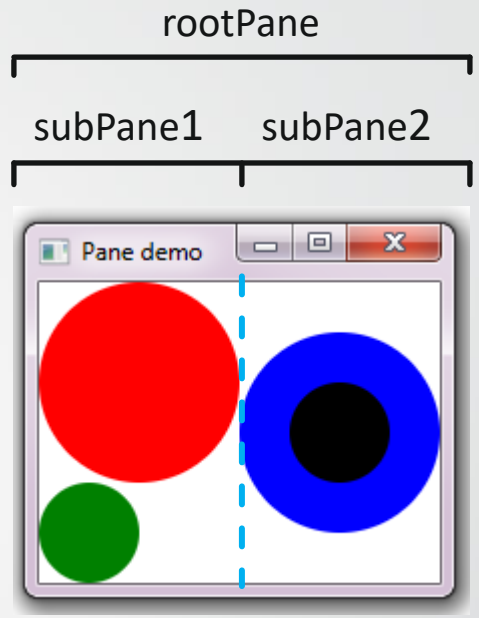


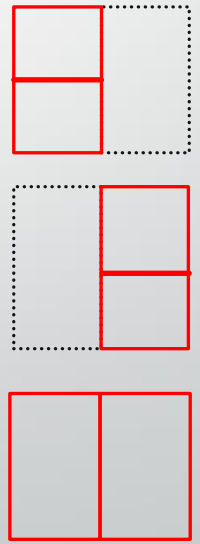
StackPane

« Layout » (4/4)

Un *Pane* peut en cacher un autre!

```
Circle myCircleRed = new Circle(50, Color.RED);
Circle myCircleGreen = new Circle(25, Color.GREEN);
Circle myCircleBlue = new Circle(50, Color.BLUE);
Circle myCircleBlack = new Circle(25, Color.BLACK);

Pane subPane1 = new VBox();
subPane1.getChildren().addAll(myCircleRed, myCircleGreen);

Pane subPane2 = new StackPane();
subPane2.getChildren().addAll(myCircleBlue, myCircleBlack);

Pane rootPane = new HBox();
rootPane.getChildren().addAll(subPane1, subPane2);
Scene myScene = new Scene (rootPane, 200, 150);
stage.setScene(myScene);
stage.setTitle("Pane demo");
stage.show();
```





Transformations

```
Rectangle rect=new Rectangle(0,0,60,60);
rect.setFill(Color.DODGERBLUE);
rect.setArcWidth(10);
rect.setArcHeight(10);
```

```
rect.setRotate(45);
```

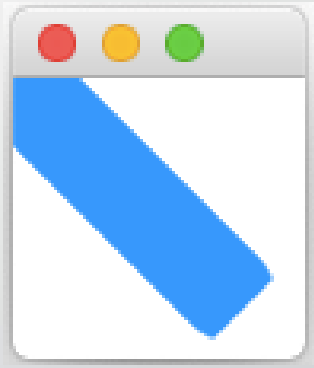
```
rect.setScaleX(2);
rect.setScaleY(0.5);
```

```
Shear shear = new Shear(0.7, 0);
rect.getTransforms().add(shear);
```

```
rect.setTranslateX(40);
rect.setTranslateY(10);
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

public class Main extends Application {
        @Override
        public void start(Stage stage) {
                try {
                        Group root = new Group();
                        Scene scene = new Scene(root,100,100);
                        stage.setScene(scene);
                        Rectangle rect = new Rectangle(0,0,60,60);
                        rect.setFill(Color.DODGERBLUE);
                        rect.setArcWidth(10);
                        rect.setArcHeight(10);

                        root.getChildren().add(rect);
                        stage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }
        public static void main(String[] args) {
                launch(args);
        }
}
```

```
rect.setRotate(45);
rect.setScaleX(2);
rect.setScaleY(0.5);
```



```
Shear shear = new Shear(0.7, 0);
rect.getTransforms().add(shear);
rect.setTranslateX(40);
rect.setTranslateY(30);
```

Transformations dans JavaFX: https://o7planning.org/fr/11157/transformations-dans-javafx

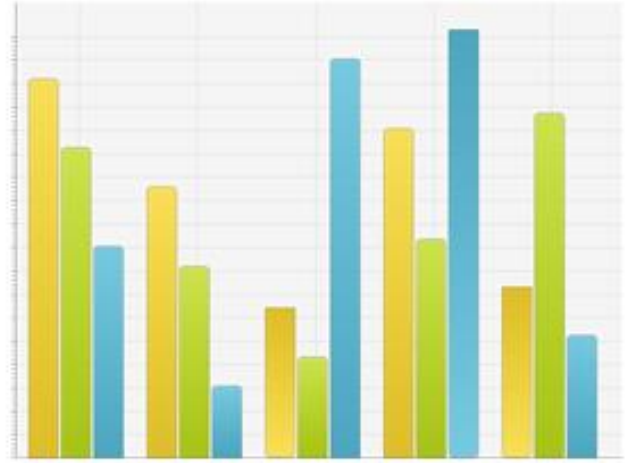
Charts

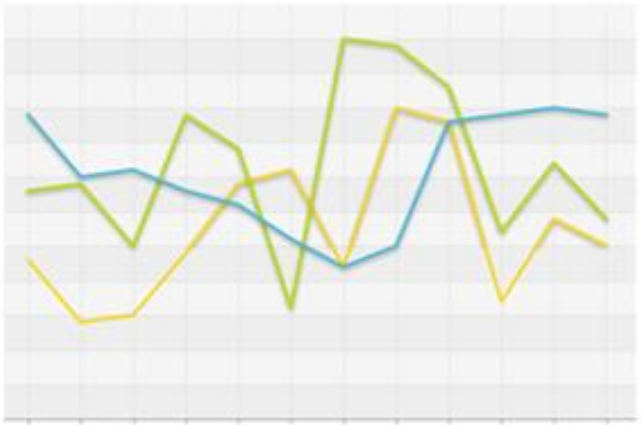
JavaFX charts is available in the **`javafx.scene.chart`** package.

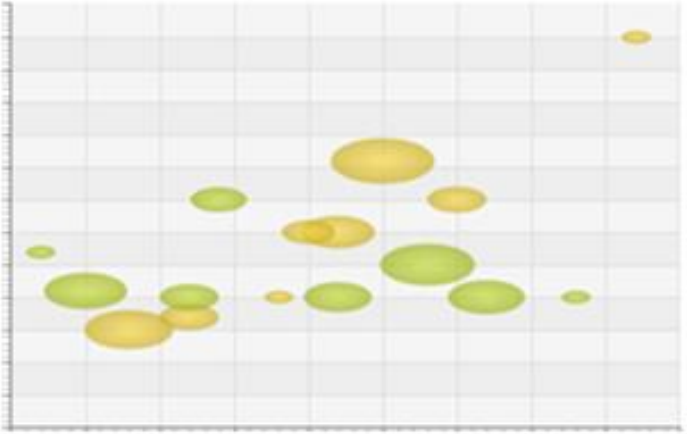
When you define the data model for a particular two-axis chart you should use the **`XYChart`** class, which provides basic capabilities for building area, line, bar, scatter, and bubble charts. Use the **`XYChart.Data`** class to specify the data model for these types of charts.
The **`xValue`** property defines the value of a chart element to be plotted on the X axis, and the **`yValue`** property defines the value for the Y axis. You can also set the extra value for each chart element. For example, to define a radius for bubble charts.

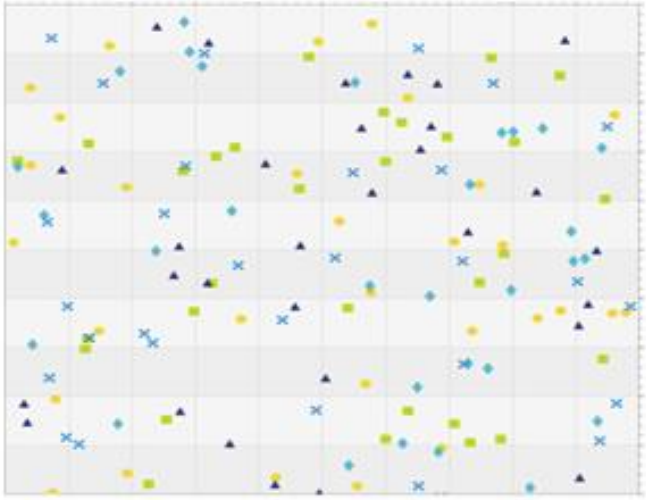
Unlike a two-axis chart, the pie chart does not require defining values for x and y axes. You use the **`PieChart.Data`** class to specify values for each slice in the pie.

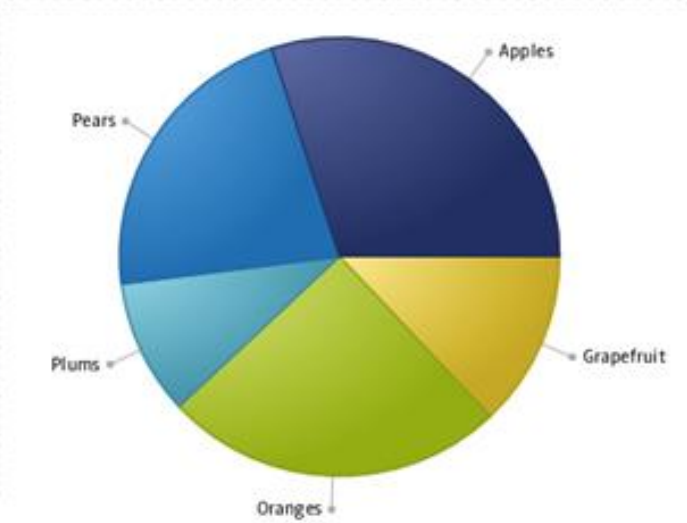
For two-axis charts, you can define several series of data by using the **`XYChart.Series`** class. For example, the line chart shown in the third figure has three series of data.







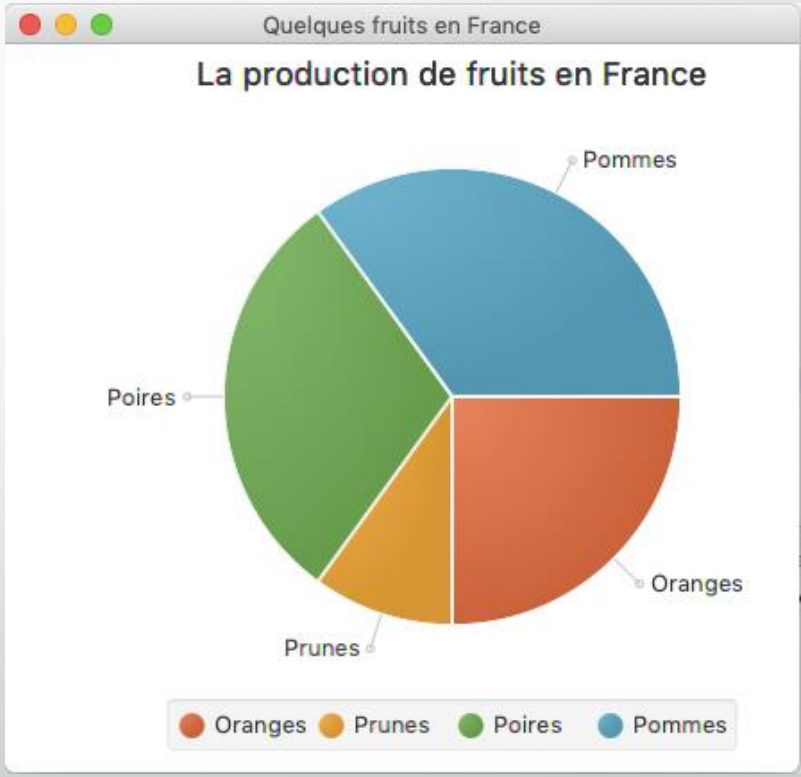




Main.java

```
public class Main extends Application {
@Override public void start(Stage stage) {
      Scene scene = new Scene(new Group());
      stage.setTitle("Quelques fruits en France");
      stage.setWidth(500);
      stage.setHeight(500);

      ObservableList<Data> pieChartData =
            FXCollections.observableArrayList(
            new PieChart.Data("Oranges", 25),
            new PieChart.Data("Prunes", 10),
            new PieChart.Data("Poires", 30),
            new PieChart.Data("Pommes", 35));
      final PieChart chart = new PieChart(pieChartData);
      chart.setTitle("La production de fruits en France");


      ((Group) scene.getRoot()).getChildren().add(chart);
      stage.setScene(scene);
      stage.show();
   }

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

Voici un exemple!



Media

Supports both visual and audio media: Media framework based on GStreamer

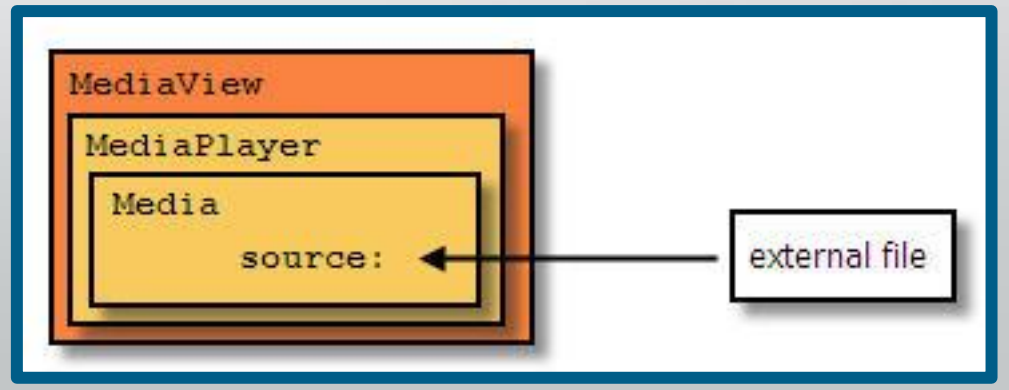
Cross platform JavaFX media file format (fxm, wav, mp3): Platform specific formats supported via native players

Features: Low latency audio, full screen support, alpha channel support

`MediaPlayer` provides control for the media rendering

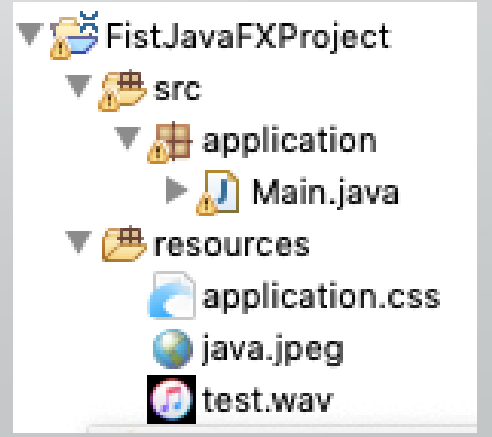
`MediaView` uses `MediaPlayer` to render media as Node

Many `MediaView` can use the same `MediaPlayer`



Media: exemple de image et audio


```
public void start(Stage stage) throws Exception {
 try {
    final URL audio = getClass().getResource("/test.wav");
    Media media = new Media(audio.toString());
    MediaPlayer player = new MediaPlayer(media);
    MediaView mediaView = new MediaView(player);

    ImageView img = new ImageView("https://variamos.com/variamosweb/img/logo.3d00fb6a.png");
    img.setX(10);
    img.setY(10);
    img.setPickOnBounds(true); // allows click on transparent areas
    img.setOnMousePressed((MouseEvent e) -> {
       player.play();
     });

     Pane myPane = new StackPane();
     myPane.getChildren().addAll(mediaView, img);
     Scene myScene = new Scene (myPane, 300, 200);
     stage.setScene(myScene);
     stage.setTitle("Cliquez sur l'image pour écouter le test.wav");
     stage.show();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```



How to Make Media Player: https://www.youtube.com/watch?v=7Gdxl2045l8

Explained example: https://docs.oracle.com/javase/8/javafx/media-tutorial/playercontrol.htm

Animated Transition

Pre-defined, single purpose animations

- Covering common use cases
- Eg. Fade, Rotate, Translate, Scale, Fill, Parallel, Sequential, Path …

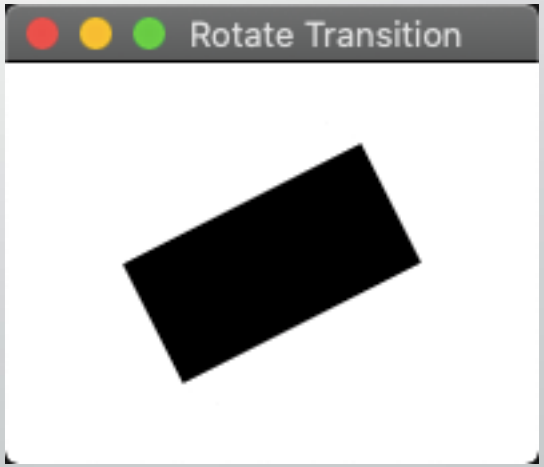
Container transition

- Parallel and sequential
- Can be nested arbitrarily
- Can be used with **`Timeline`**

More about transitions and animations:
https://www.youtube.com/watch?v=jkYKoRX8i3Q
https://docs.oracle.com/javafx/2/animations/basics.htm
https://docs.oracle.com/javafx/2/animations/tree-animation.htm

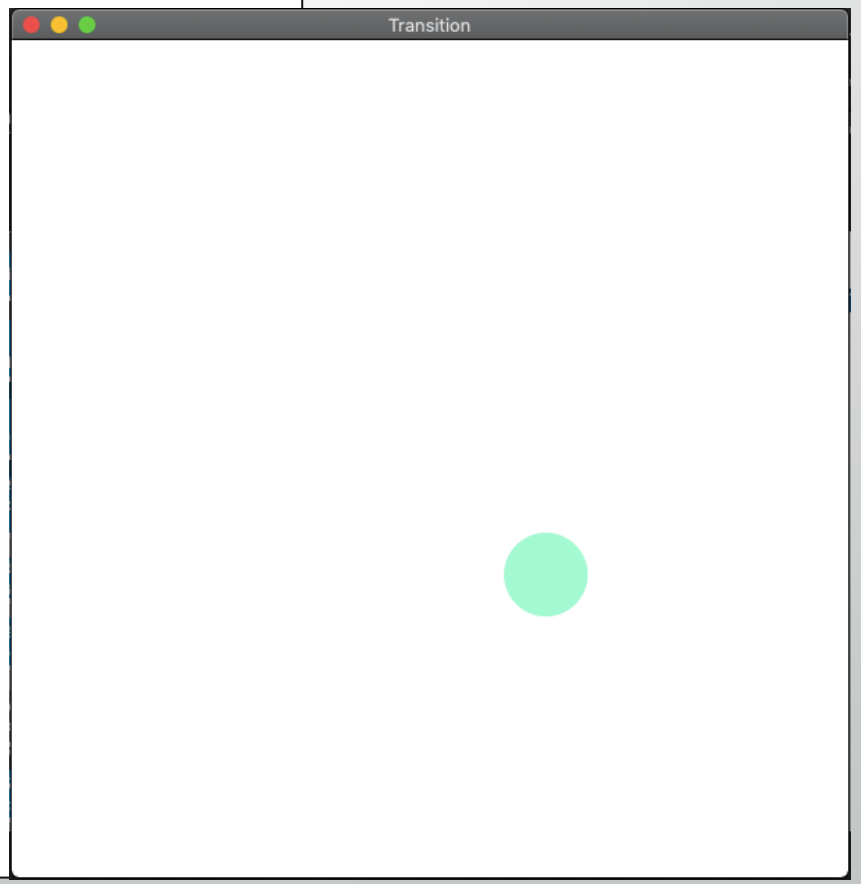
Rotate transition

```
public class App extends Application {
    @Override
    public void start(Stage stage) {
        Rectangle rect = new Rectangle(100, 50);
        rect.setLayoutX(50);
        rect.setLayoutY(50);
        RotateTransition rt = new RotateTransition(Duration.millis(9000), rect);
        //use setNode (rt.setNode(rect);) to be able to reuse the transitions
        rt.setByAngle(180); //how many degrees do you want to rotate
        rt.play();

        Pane root = new Pane();
        root.getChildren().add(rect);
        Scene scene =  new Scene(root, 300, 300);
        stage.setTitle("Rotate Transition");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```



Translate transition

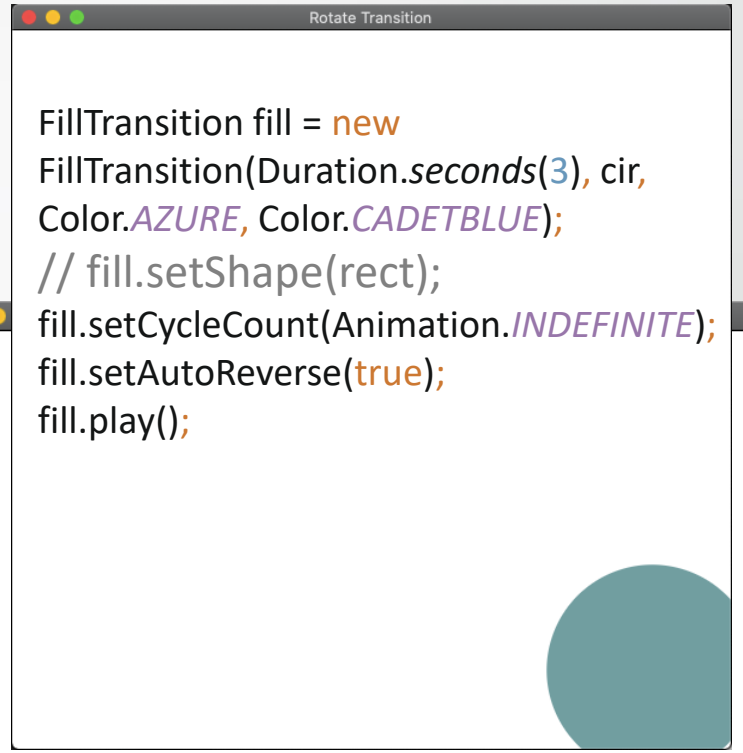
```
public class App extends Application {
    @Override
    public void start(Stage stage) {
        Circle cir = new Circle();
        cir.setFill(Color.AQUAMARINE);
        cir.setRadius(30);
        cir.setLayoutX(50);
        cir.setLayoutY(50);

        TranslateTransition translate =  new TranslateTransition(Duration.seconds(3), cir);
        translate.setToX(500);
        translate.setToY(500);
        translate.setAutoReverse(true);
        translate.setCycleCount(Animation.INDEFINITE);
        translate.play();

        Pane root = new Pane();
        root.getChildren().add(cir);
        Scene scene =  new Scene(root, 600, 600);
        stage.setTitle("Rotate Transition");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```



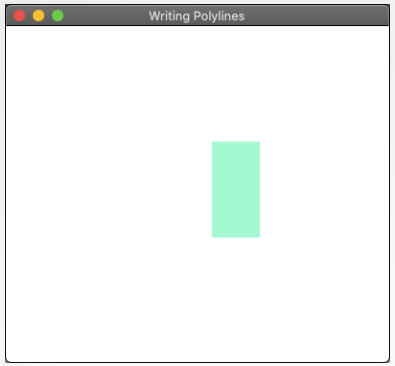
```
public class App extends Application {
    @Override
    public void start(Stage stage) {
        Circle cir = new Circle();
        cir.setFill(Color.AQUAMARINE);
        cir.setRadius(30);
        cir.setLayoutX(50);
        cir.setLayoutY(50);

        TranslateTransition translate =  new TranslateTransition(Duration.seconds(3), cir);
        translate.setToX(500);
        translate.setToY(500);
        translate.setAutoReverse(true);
        translate.setCycleCount(Animation.INDEFINITE);
        translate.play();

        ScaleTransition scale =  new ScaleTransition(Duration.seconds(3), cir);
        scale.setCycleCount(Animation.INDEFINITE);
        scale.setAutoReverse(true); //What behaviour if you put false here?
        scale.setToX(3);
        scale.setToY(3);
        scale.play();

        Pane root = new Pane();
        root.getChildren().add(cir);
        Scene scene =  new Scene(root, 600, 600);
        stage.setTitle("Rotate Transition");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

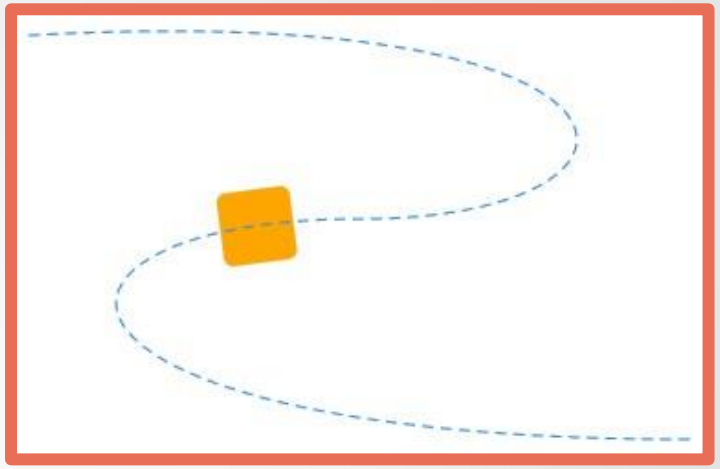
Follow this tutorial for more examples:
https://www.javatpoint.com/javafx-fill-transition

Translate & Scale transitions



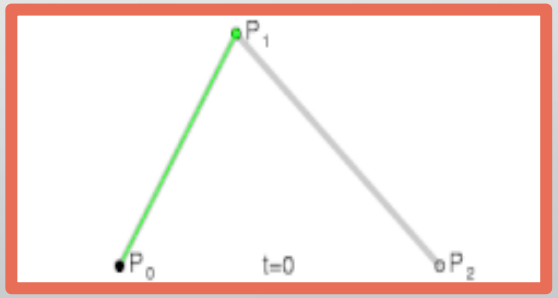
```
public class App extends Application {
    @Override
    public void start(Stage stage) {
        Rectangle rect = new Rectangle(50, 100);
        rect.setFill(Color.AQUAMARINE);
        rect.setLayoutX(50);
        rect.setLayoutY(50);

        Path path = new Path();
        path.getElements().add(new MoveTo(0, 0));
        path.getElements().add(new CubicCurveTo(300, 0, 300, 120, 200, 120));
        path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 300, 240));
        PathTransition pathTransition =  new PathTransition();
        pathTransition.setNode(rect);
        pathTransition.setDuration(Duration.seconds(3));
        pathTransition.setPath(path);
        pathTransition.setCycleCount(PathTransition.INDEFINITE);
        pathTransition.play();

        Pane root = new Pane();
        root.getChildren().add(rect);
        Scene scene =  new Scene(root, 400, 350);
        stage.setTitle("Writing Polylines");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```





https://docs.oracle.com/javafx/2/animations/basics.htm

CubicCurveTo(double controlX1, double controlY1, double controlX2, double controlY2, double x, double y)



Creates a curved path element, defined by three new points, by drawing a Cubic Bézier curve that intersects both the current coordinates and the specified coordinates (x,y), using the specified points (controlX1, controlY1) and (controlX2, controlY2) as Bézier control points. All coordinates are specified in double precision.

More about the Bézier curves: https://en.wikipedia.org/wiki/B%C3%A9zier_curve

Quatre pièges à éviter …

1. [avec Java FX] Typage, attention aux imports
2. [avec Java FX] “*Toolkit not initialized*”
3. [avec Java] Attributs et variables locales
4. [avec Eclipse] Classes JavaFX invisibles

Quatre pièges à éviter …

1. [avec Java FX] Typage, attention aux imports

Attention aux imports: votre IDE vous proposera plusieurs options mails il faut choisir les objets de la librairie JavaFX

```
@Override
public void start(Stage primaryStage) throws Exception {
    Label myLabel = new Label("Hello World");

}

stati
    l
}
```

Class to Import

- Label (javafx.scene.control) < 1.8.0.121 > (jfxrt.jar)
- Label (jdk.internal.org.objectweb.asm) < 1.8.0.121 > (rt.jar)
- Label (jdk.nashorn.internal.codegen) < 1.8.0.121 > (nashorn.jar)
- Label in Decoration (sun.font) < 1.8.0.121 > (rt.jar)

Quatre pièges à éviter …

2. [avec Java FX] “Toolkit not initialized”

Exception in thread "main" java.lang.IllegalStateException:

Toolkit not initialized

On ne peut pas utiliser une librairie (JavaFX) qui n’a pas été encore initialisée !

Pour cette raison on **ne peut pas utiliser ni le constructeur ni le « *main* » pour initialiser les éléments graphiques!**

Quatre pièges à éviter …

3. [avec Java] Attributs et variables locales

Lorsque vous déclarez des attributs, attention aux ambiguïtés avec les variables locales:

```
public class HelloWorld extends Application {

    private Label myLabel;

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label myLabel = new Label( text: "Hello World!");
```

Quatre pièges à éviter …

3. [avec Java] Attributs et variables locales

Lorsque vous déclarez des attributs, attention aux ambiguïtés avec les variables locales:

```
public class HelloWorld extends Application {

    private Label myLabel;

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label myLabel = new Label( text: "Hello World!");
```

/!\ Déclaration d'une nouvelle variable locale /!

myLabel (local) et ***this.myLabel*** sont alors deux *Label* différents

Cinq pièges à éviter …

3. [avec Java] Attributs et variables locales

Lorsque vous déclarez des attributs, attention aux ambiguïtés avec les variables locales:

```
public class HelloWorld extends Application {

    private Label myLabel;

    @Override
    public void start(Stage primaryStage) throws Exception {

Correct!    myLabel = new Label( text: "Hello World!");
```

Pour éviter les ambiguïtés:

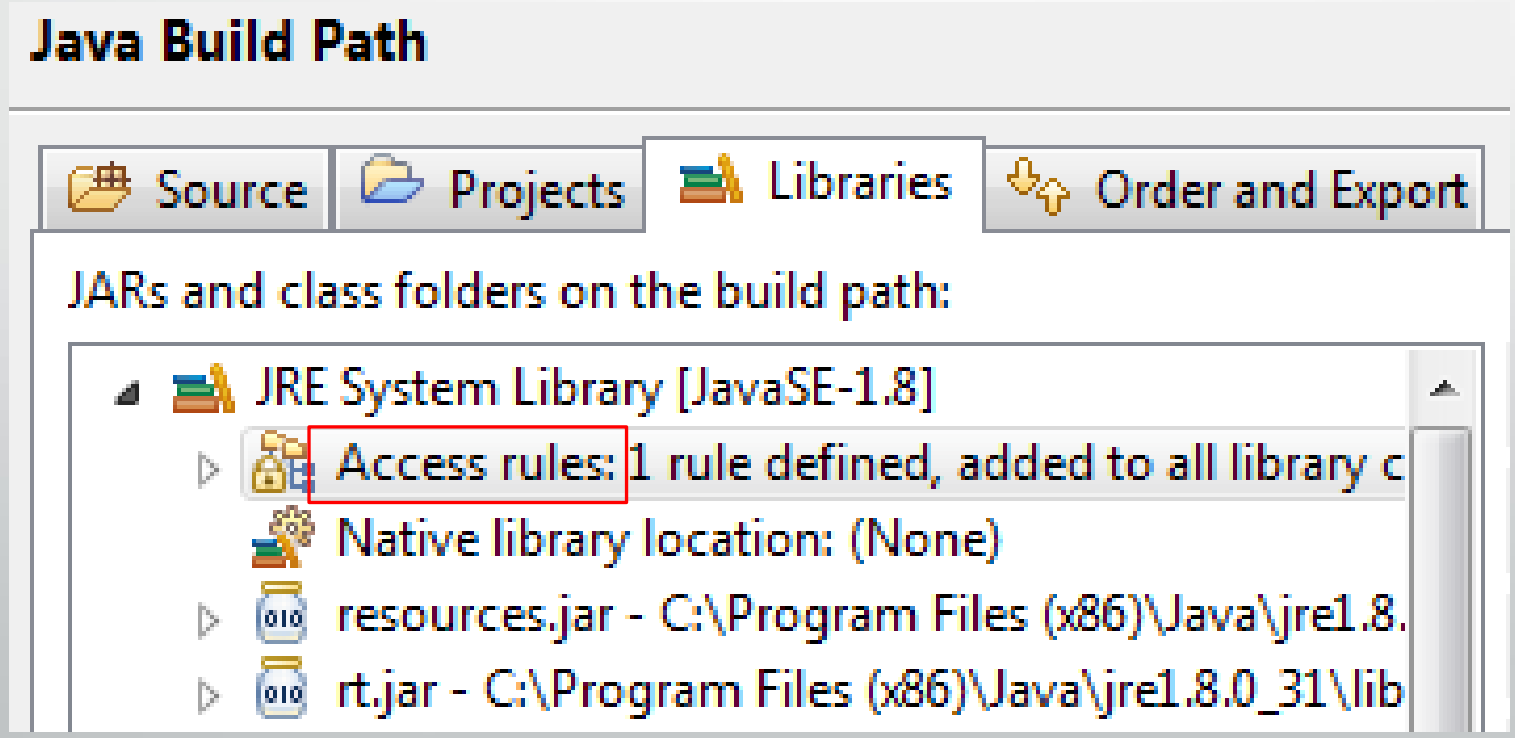
```
        this.myLabel = new Label( text: "Hello World!");
```

Quatre pièges à éviter …

4. [avec Eclipse] Classes JavaFX invisibles

Vous ne trouvez pas les classes Java FX ? **Eclipse** les « cache » par défaut.

Naviguez jusqu'aux règles d'accès (Build Path > Librairies > JRE System Library > Access rules

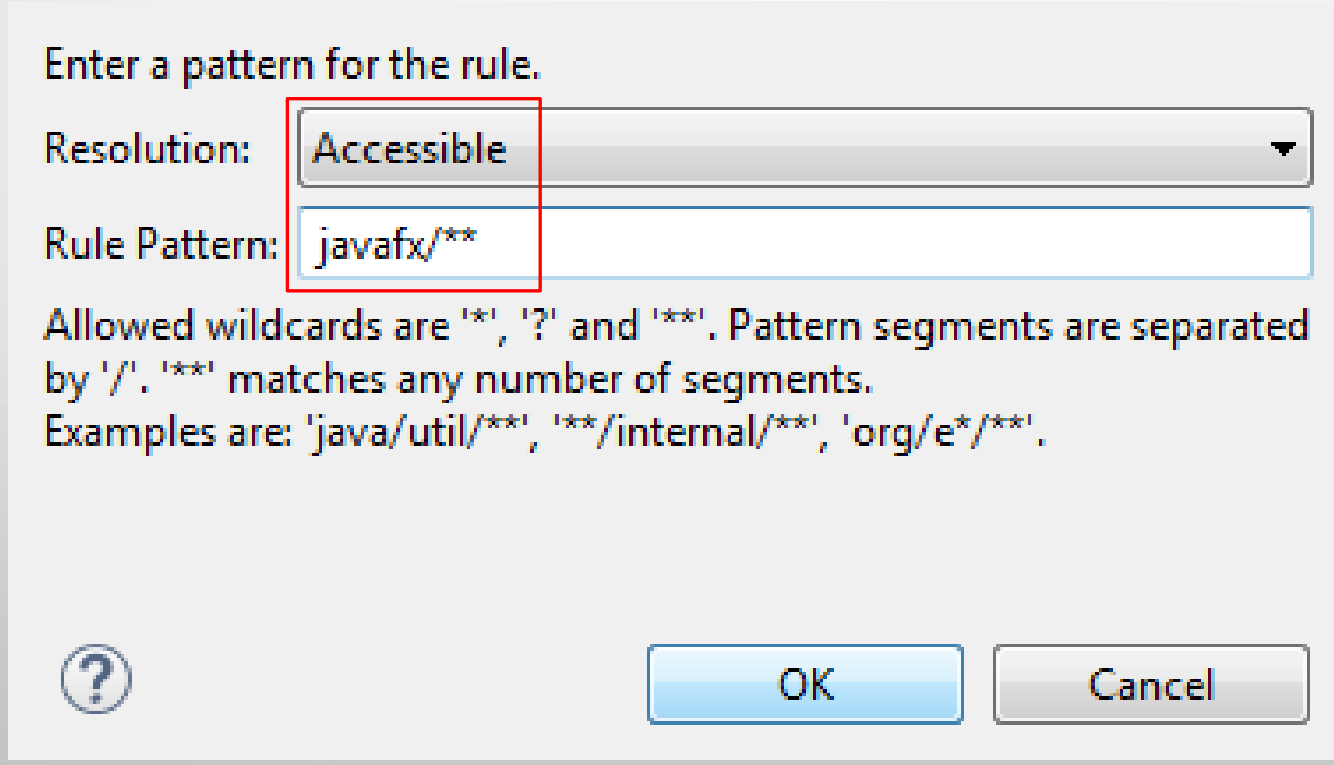


Quatre pièges à éviter …

4. [avec Eclipse] Classes JavaFX invisibles

Vous ne trouvez pas les classes Java FX? Eclipse les « cache » par défaut.

Cliquez « *Add* » (ou « Edit » si déjà présente mais incorrecte) > package javafx visible:



```
public class App extends Application {
    private Polyline polyline;

    @Override
    public void start(Stage stage) {
        AnchorPane root = new AnchorPane();
        Scene scene = new Scene(root, 500, 500);

        scene.setOnMousePressed(event -> {
            polyline = new Polyline();
            root.getChildren().add(polyline);
        });
        scene.setOnMouseDragged(event -> {
            polyline.getPoints().addAll(event.getX(), event.getY());
        });

        stage.setTitle("Writing Polylines");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

To play with more examples, try this tutorial: https://www.javatpoint.com/javafx-tutorial

C'est la fin de ce cours!

Lundi prochain matin (**Hiba et Raúl: sous RDV) Séance de tutorat et pré-présentation (optionnelle) des projets**

Dimanche 7 janvier (au plus tard): envoie des projets par le canal privé (Teams) de chaque équipe

10 et 11 janvier 08:10 - 12:15 (**Hiba et Raúl) Présentation des projets et feedback/notes**

This is the end of this course! But for you…

it is just the beginning!

All the best!

Raúl Mazo