Première application

Contenu d'une application

Les différents éléments nécessaires

- Classes Java
- Fichiers FXML
- Fichiers properties
- Fichiers CSS
- JVM 7 ou supérieur

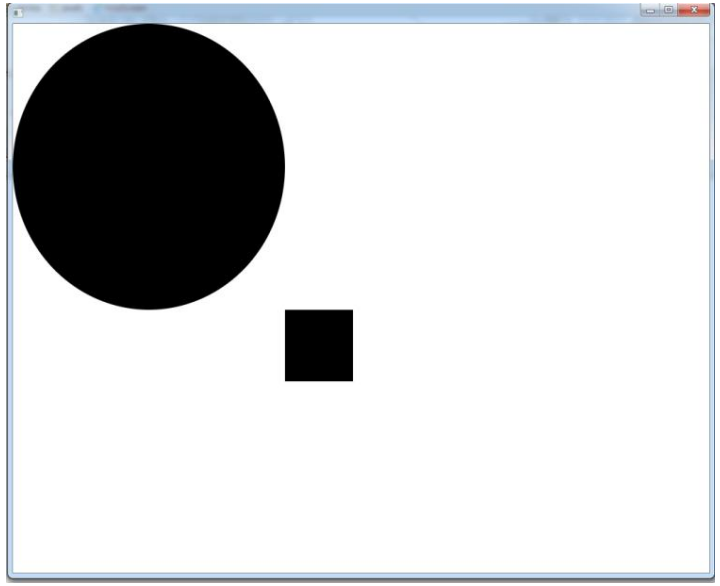
BOOTSTRAP

Démarrer avec JavaFX

- Une application JavaFX doit étendre la classe javafx.Application
- Le démarrage se fait via une méthode statique nommée « launch »
  - Initialisation de divers paramètres propres à JavaFX
  - La méthode launch utilise le thread courant comme UI Thread
- La fenêtre est représentée par un objet Stage
  - Plusieurs Stages peuvent être gérés (popups, multi-écrans…)
- Une Scene est affichée
  - Contient l'arborescence des composants qui sont des Nodes

EXEMPLE D'APPLICATION

```
public class FirstScreen extends Application {
    @Override
    public void start(final Stage stage) throws Exception {
        Group group = new Group();

        Circle circle = new Circle();
        circle.setRadius(200);
        circle.setCenterX(200);
        circle.setCenterY(200);
        group.getChildren().add(circle);

        Rectangle rectangle = new Rectangle();
        rectangle.setHeight(100);
        rectangle.setWidth(100);
        rectangle.setX(400);
        rectangle.setY(400);
        group.getChildren().add(rectangle);

        final Scene scene = new Scene(group, 1024, 768);
        stage.setScene(scene);
        stage.show();
    }

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

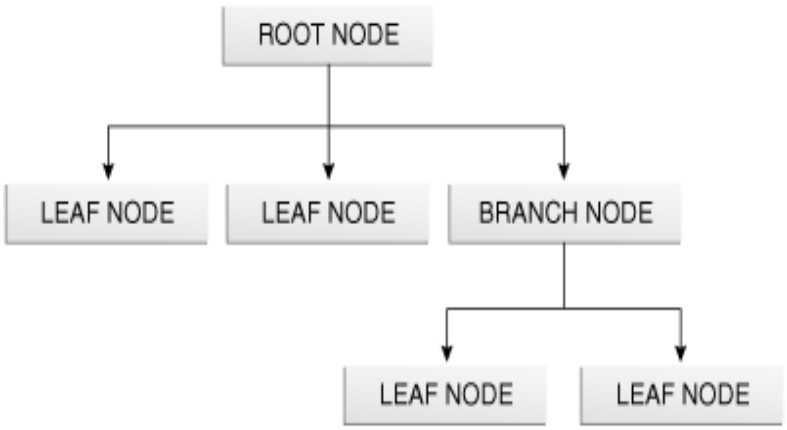


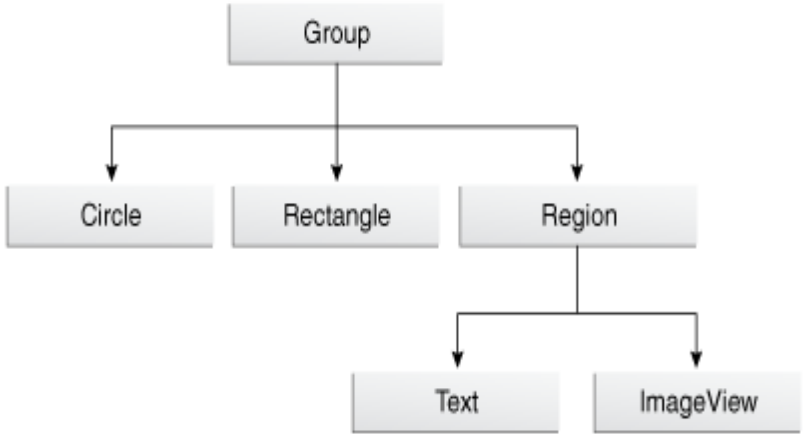
Le Scene Graph

Définir les éléments de la vue

- Arbre de composants JavaFX avec un composant racine
- Décrit le contenu d'une Scene
- Chaque élément de l'arbre est nommé « Node »
- Chaque Node peut avoir des enfants (pattern composite)

- Exemple :





Définir un écran avec FXML

Une abstraction intéressante

- Langage déclaratif de construction d'interface basé sur du XML
- Décrit un arbre de composants (avec une racine unique)
  - Un fichier FXML peut être chargé par la Scene
- Relié au code par un Controller
  - Possibilité d'encapsuler des scripts (peu utilisé et peu recommandé)
  - Intégration de type MVC ou MVP
- Peut être conçu visuellement à l'aide de SceneBuilder (WYSIWYG)

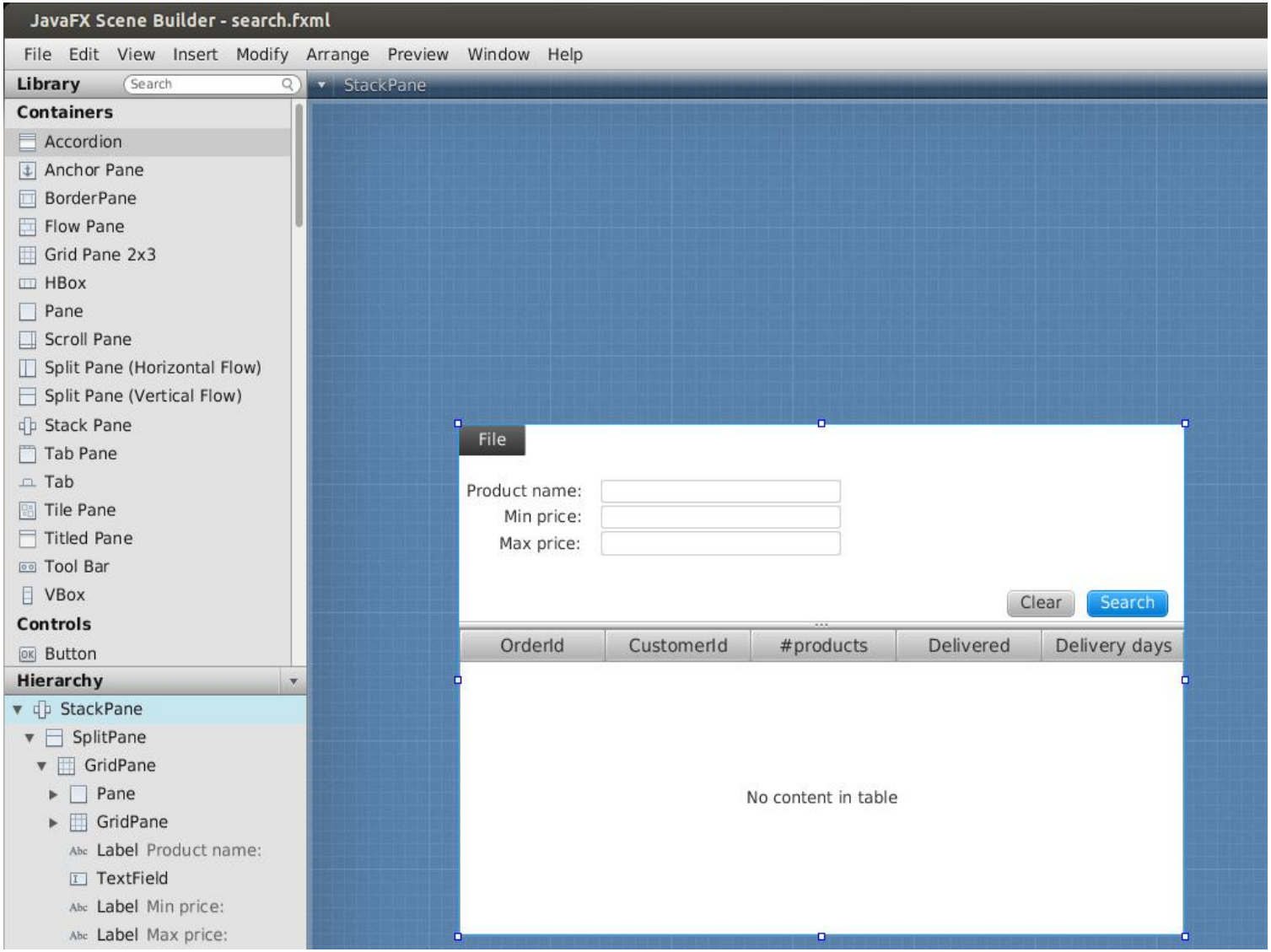
EXEMPLE D'UTILISATION

GridPane avec layout en rows / columns

```
<GridPane fx:controller="fxmlexample.FXMLExampleController"
xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">
    <padding><Insets top="25" right="25" bottom="10" left="25"/></padding>
    <Text text="Welcome"
        GridPane.columnIndex="0" GridPane.rowIndex="0"
        GridPane.columnSpan="2"/>

    <Label text="User Name:"
        GridPane.columnIndex="0" GridPane.rowIndex="1"/>

    <TextField
        GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <Label text="Password:"
        GridPane.columnIndex="0" GridPane.rowIndex="2"/>

    <PasswordField fx:id="passwordField"
        GridPane.columnIndex="1" GridPane.rowIndex="2"/>

     <Button fx:id="okButton" GridPane.columnSpan="2"
          GridPane.columnIndex="0" GridPane.rowIndex="3" />
</GridPane>
```

L'OUTIL SCENE BUILDER

What You See Is What You Get (WYSIWYG)



POURQUOI UTILISER FXML?

Plusieurs avantages

- Découpler la logique d'écran de sa déclaration
  - Meilleure maintenabilité
  - Meilleure évolutivité
  - Possibilité de découpage de travail avec un graphiste
- Le FXML est central dans la mise en place de design patterns de type MVC ou MVP
  - Intégration avec des frameworks Java
- Le FXML est évalué : il n'est pas compilé, on peut donc générer dynamiquement des vues via ce format XML

Chargement d'un FXML

Utilisation dans une application

- FXMLLoader est dédié au chargement des fichiers FXML et leur transformation en arbre de composants

```
//dans la méthode start, au lieu d'instancier directement root
URL location = getClass().getResource("zenfxapp.fxml");
FXMLLoader fxmlLoader = new FXMLLoader(location);
BorderPane root = (BorderPane) fxmlLoader.load();

//on peut aussi récupérer une instance du contrôleur
MyController controller = (MyController)fxmlLoader.getController();

//reste du chargement de la scène
Scene scene = new Scene(root,400,400);
```

- Ou utiliser FXMLLoader ?
  - Soit lors de l'instanciation d'un composant déclaré via FXML dans le composant parent
  - Soit dans le constructeur d'un composant qui se décrit en FXML

PLUSIEURS UTILISATIONS POSSIBLES

FXML First ou Java First?

- Plusieurs découpages d'application sont possibles avec FXML
  - 1 FXML = 1 « page » de l'application
  - 1 FXML = 1 composant de l'application (plusieurs composants par page)
- Un fichier FXML peut charger un autre fichier FXML via la balise <fxml:include>
- Deux choix sont possibles pour les chargements en cascade de FXML
  - Les fichiers FXML explicitent les fichiers enfants à charger (fxml:include) : FXML First
  - Les contrôleurs Java définissent les FXML enfants à charger : Java First

NOTION DE CONTRÔLEUR

Le pattern MVC selon Oracle

- Le comportement associé au FXML est défini dans une classe "Controller"
  - Il n'est pas nécessaire d'implémenter une interface / d'hériter d'une classe
  - Interactions FXML / Controller via des listeners d'événements ou des méthodes annotées
- JavaFX effectue la liaison FXML / Controller via réflexion sur la classe Java
  - Le nommage des méthodes / champs est important
  - Utilisation d'annotations propres à JavaFX comme @FXML
  - Impact sur la performance lors de chargements de nombreux FXML
- Il est possible de récupérer des paramètres d'exécution dans le Controller
  - Exemple : un événement en paramètre

```
public void maMethode(TotoEvent event) { … }
```

Exemple d'interactions

Lier FXML et contrôleur

```
<GridPane fx:controller="fxmlexample.FXMLExampleController"
          xmlns:fx="http://javafx.com/fxml" alignment="center"
hgap="10" vgap="10">

          <Button fx:id="okButton" text="Ok"
              onAction="#okButtonHandler" />

</GridPane>
```

Exemple.fxml

```
public class FXMLExampleController {

          public void okButtonHandler(ActionEvent event) {
           System.out.println("ZenOk !");
          }
}
```

FXMLExampleController.java

- Un appel de méthode est présent dans le FXML via #
- La méthode en question est codée dans le Controller
- Elle prend un ActionEvent en paramètre

Exemple d'interactions

Une autre technique possible

- Il est possible d'injecter des composants dans un Controller
  - Soit par son nom seul en scope public
  - Soit par l'annotation @FXML
  - Il est recommandé de toujours définir l'annotation
- En implémentant l'interface Initializable de JavaFX
  - La méthode initialize est appelée par JavaFX à la fin de la construction de la vue
  - On peut donc ajouter des eventHandler en Java

Exemple d'interactions

Une autre technique possible

```
public class FXMLExampleController implements Initializable {

    @FXML private Button okButton;

    public void initialize(URL location, Resources ressources) {
        okButton.setOnAction(new EventHandler<ActionEvent>() {

            public void handle(ActionEvent event) {
                System.out.println("ZenOk !");
            }

        });
    }

}
```

Construction par code Java

L'alternative procédurale

- Construction sans FXML, "comme en Swing"

```
final VBox rootNode = new VBox();
rootNode.getChildren().add(new Label("Hello
World !"));
```

- Spécificités d'un développement purement procédural
  - Dynamique de l'UI plus facilement gérable en Java
  - Pas d'interprétation au runtime du rendu (code entièrement compilé)
  - Accès à des API de bas niveau (exemple : écrire des pixels à une certaine position)
  - Meilleure performance
  - Transfert de compétences aisé pour les développeurs Swing / Eclipse RCP

FXML ou Java?

Procédural versus déclaratif

- Le FXML permet de mieux séparer le code
  - XML pour la vue statique
  - Controller Java pour les interactions et la dynamique
- Le FXML a quelques défauts
  - Coût en performance
  - Certaines interactions ne peuvent être décrites en déclaratif

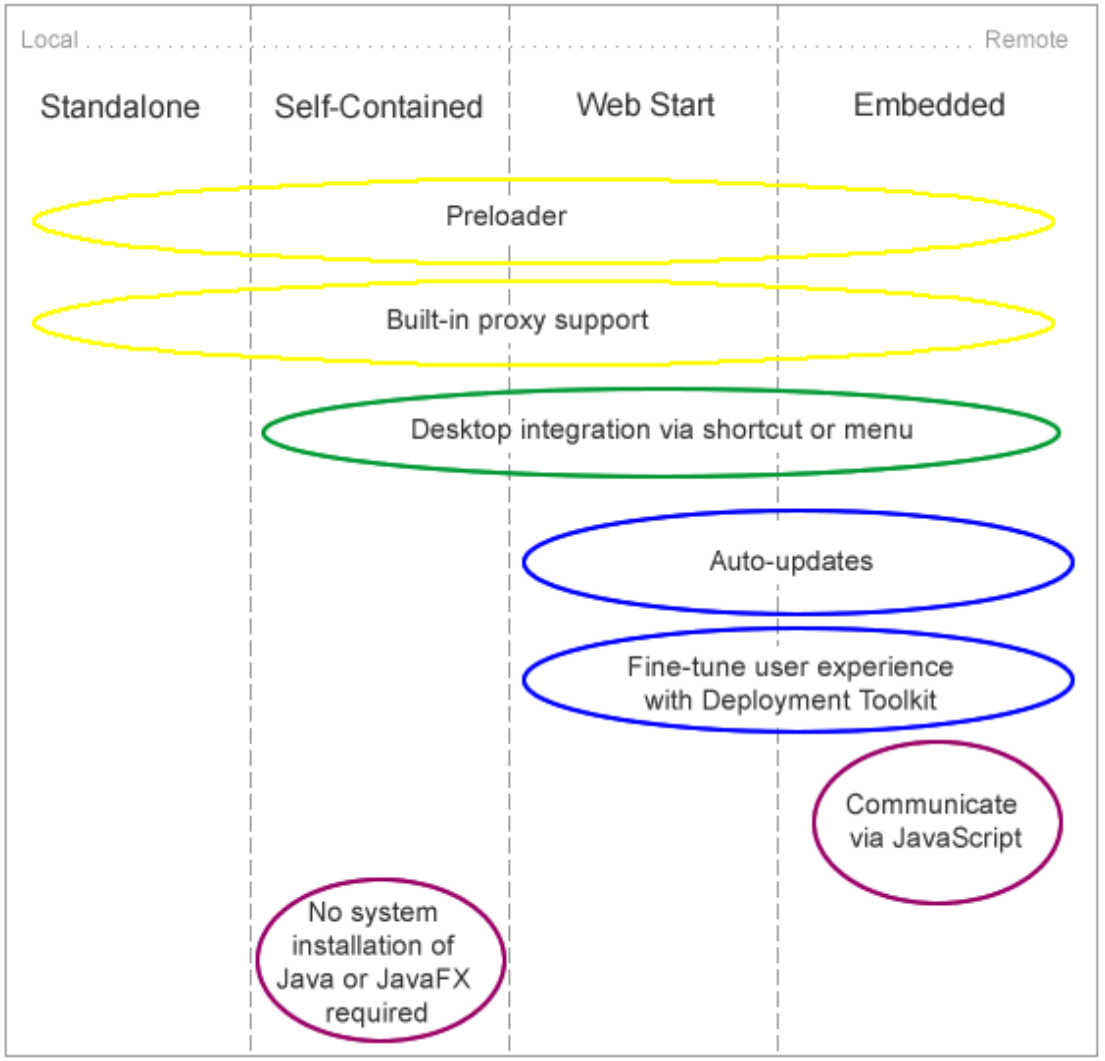
La majorité des vues devraient donc, dans un projet, être en FXML

Quelques éléments purs Java complexes peuvent aussi être définis

LES DÉPLOIEMENTS

Plusieurs options possibles

- 4 modes disponibles avec des livrables différents
- Gestion des mises à jour applicatives sous conditions
- Possibilité de construire un installateur suivant l'OS cible

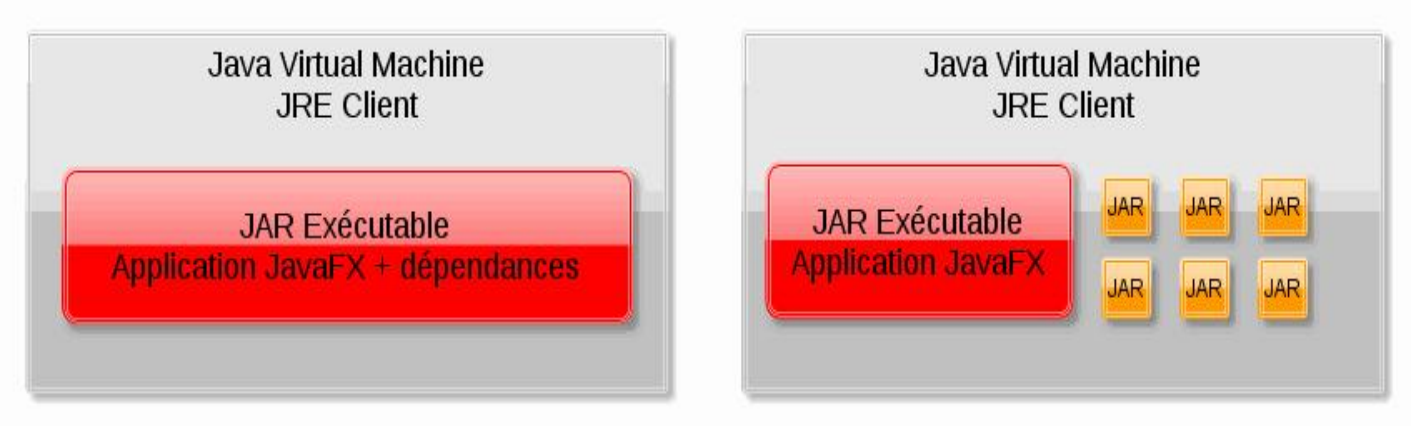


SHIP IT!

- Le mode Self-contained
  - Création d'un installateur standard pour un OS donné
  - Inclut le JRE nécessaire
  - Livrable volumineux mais standard pour l'OS
  - Peu adapté pour des applications mises à jour fréquemment
- Le mode Embedded
  - Pour lancer l'application depuis un navigateur Web
  - Facilite les échanges JavaFX / JavaScript sur une page Web
  - Nécessite un JRE
  - Pas adapté pour une application déconnectée

SHIP IT!

- Mode Standalone
  - Livraison d'un JAR exécutable par un JRE
  - Package de l'ensemble des dépendances dans un seul JAR
  - Package des classes applicatives dans un seul JAR + livraison des dépendances JAR externes



- Le mode WebStart
  - Déploiement sur un serveur Web pour gestion automatique des versions
  - Permet toutefois le lancement de l'application en mode déconnecté