

JavaFX & FXML

martine.gautier@univ-lorraine.fr

▲ Découpler la description de l'IG du reste du code

▲ Intérêts

- L'arborescence des composants est facilement identifiable, donc modifiable.
- La description de l'IG est indépendante du langage de programmation.
- La description des composants (placement, apparence) est plus facile que la programmation (on compte 106 setter dans la classe Button ...)
- Intégrer la localisation est simple.



La séparation entre modèle et IG subsiste.

▲ Fichier au format FXML en 3 parties

- Entête

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

- Imports (Java like)

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

- Balises

```
<AnchorPane prefHeight="264.0" . . . >
```

- ▲ Les balises portent le nom d'un composant graphique.
 - AnchorPane, BorderPane, Button, Image, ... etc.

▲ Description de l'arborescence

```
<AnchorPane . . . >  
  <children>  
    <Button . . . >  
    <Label . . . >  
  </children>  
</AnchorPane>
```

Définition d'un AnchorPane avec deux composants Button et Label

▲ Les attributs définissent les propriétés des composants.

- Texte

```
text="Nouveau"
```

- Placement

```
layoutX="152.0" layoutY="232.0"
```

```
GridPane.columnIndex="1"
```

- Tailles (préférées)

```
prefHeight="220.0" prefWidth="140.0"
```

- Style

```
style="-fx-background-color: white;"
```

- CSS

```
stylesheets="@../style.css"
```

▲ Les attributs définissent les propriétés des composants.

- Texte

`text="Nouveau"`

- Placement

`layoutX="152.0" layoutY="152.0"`

`GridPane.columnIndex=0`

- Tailles (préférées)

`prefHeight="220.0" prefWidth="220.0"`

- Style

`style="-fx-background-color: white;"`

- CSS

`stylesheets="@../style.css"`

@ désigne le répertoire qui contient le fichier dans lequel on écrit cet attribut

- Image

```
<ImageView fitWidth="18.0" preserveRatio="true">  
  <image>  
    <Image url="@../resources/yellowDuck.png" />  
  </image>  
</ImageView>
```

- ... impossible de les passer tous en revue

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

- Création d'un projet JavaFX dans IntelliJ
 - = Répertoire **sample** avec 3 fichiers
 - Main.java sample.xml Controller.java
 - = Le fichier Controller.java est vide.
 - = Dans Main.java
 - Référence à sample.xml dans la fonction start
 - Aucune création de composant graphique
- Exécution
 - = Fenêtre HelloWorld

Trois solutions possibles

Au choix

1 = ajouter des classes définissant les vues ; instancier ces vues dans la fonction start de Main.java

→ on n'utilise ni le fichier xml, ni le contrôleur

2 = compléter le fichier sample.xml avec des balises décrivant les composants

3 = ouvrir sample.xml avec SceneBuilder pour le modifier

Trois solutions possibles

1 = ajouter des classes définissant les vues ; instancier ces vues dans la fonction start de Main.java

Cf. 2048

2 = compléter le fichier sample.xml avec des balises décrivant les composants

3 = ouvrir sample.xml avec SceneBuilder pour le modifier

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

Cf. Tp à venir

```
public void start(Stage primaryStage) ...
```

```
    Parent root =
```

```
    FXMLLoader.load(getClass().getResource("sample.fxml"));
```

```
    primaryStage.setScene(new Scene(root, 300, 275));
```

La fonction FXMLLoader.load

- charge le fichier FXML,
- instancie une arborescence de composants graphiques avec les propriétés spécifiées ;
- le type de l'instance créée est défini par la racine de l'arborescence (GridPane, ici).

▲ Modification du fichier `sample.xml`

- avec SceneBuilder (clic droit sur `sample.xml`)
- en mode texte

ET/OU

Ajouter des
composants

▲ Modification du fichier sample1

- avec SceneBuilder (clic droit sur le fichier)

ET/OU

- en mode

Attention, dans certains
environnements,
SceneBuilder ne fonctionne
pas dans IntelliJ. Il faut le
télécharger et l'utiliser à
côté.

▲ Modification du fichier `sample.xml`

- avec SceneBuilder (clic droit sur `sample.xml`)
- en mode texte

ET/OU

▲ Fenêtre principale de SceneBuilder

- vue graphique du composant (centre)
- Library : tous les composants JavaFX possibles
- Document : arborescence des composants
- Inspector : attributs d'un composant particulier

▲ Fenêtre Inspector

- Onglet Properties : propriétés du composant
texte, couleur, transparence, alignement, css, etc.
- Onglet Layout : placement/taille/échelle
- Onglet Code : réactivité
onAction, onMouseClicked, onDragDetected, onZoom, etc.

▲ Identifier les événements que l'IG doit prendre en compte

▲ Attacher une fonction à chacun d'eux

```
<Button    mnemonicParsing="false"  
          onAction="#reagirAuClic"  
          text="JeSuisUnJoliBouton"  
/>
```

▲ Ecrire la fonction **reagirAuClic** ... mais où ?

- ▲ Le contrôleur entre en scène.
- ▲ Instance d'une classe publique avec constructeur public sans paramètre
 - attachée à un composant graphique
 - créée au chargement du fichier .xml
 - permet de manipuler les entités décrites dans le .xml
 - contient les fonctions gérant la réactivité

```
<GridPane . . . fx:controller="sample.Controller">  
  
    <Button    mnemonicParsing="false"  
              onAction="#reagirAuClic"  
              text="JeSuisUnJoliBouton" />  
  
</GridPane>
```

```
package sample ;  
class Controller {  
  
    public Controller() {}  
  
    public void reagirAuClic() {  
        System.out.println("Coucou c'est moi"); }  
}
```

```
package sample ;  
class Controller {
```

```
    public Controller() {}
```

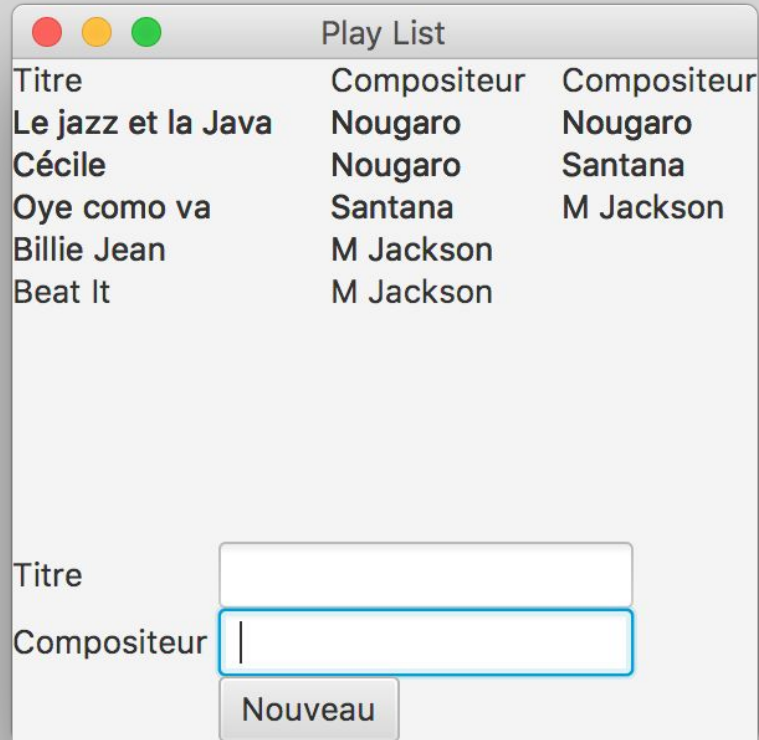
```
    @FXML
```

```
    void reagirAuClic() {  
        System.out.println("Coucou c'est moi"); }  
}
```

Alternative : la fonction
est déclarée public ou
bien sa définition est
précédée de
l'annotation FXML

- ▲ Cet exemple est rudimentaire.
- ▲ Apprendre à
 - créer un composant autonome pour chaque vue
 - donner à chaque composant l'accès au modèle
 - donner à chaque composant l'accès à ses constituants

PlayList



Titre	Compositeur	Compositeur
Le jazz et la Java	Nougaro	Nougaro
Cécile	Nougaro	Santana
Oye como va	Santana	M Jackson
Billie Jean	M Jackson	
Beat It	M Jackson	

Titre

Compositeur

Nouveau

Play List

Titre	Compositeur
Le jazz et la Java	Nougaro
Cécile	Nougaro
Oye como va	Santana
Billie Jean	M Jackson
Beat It	M Jackson

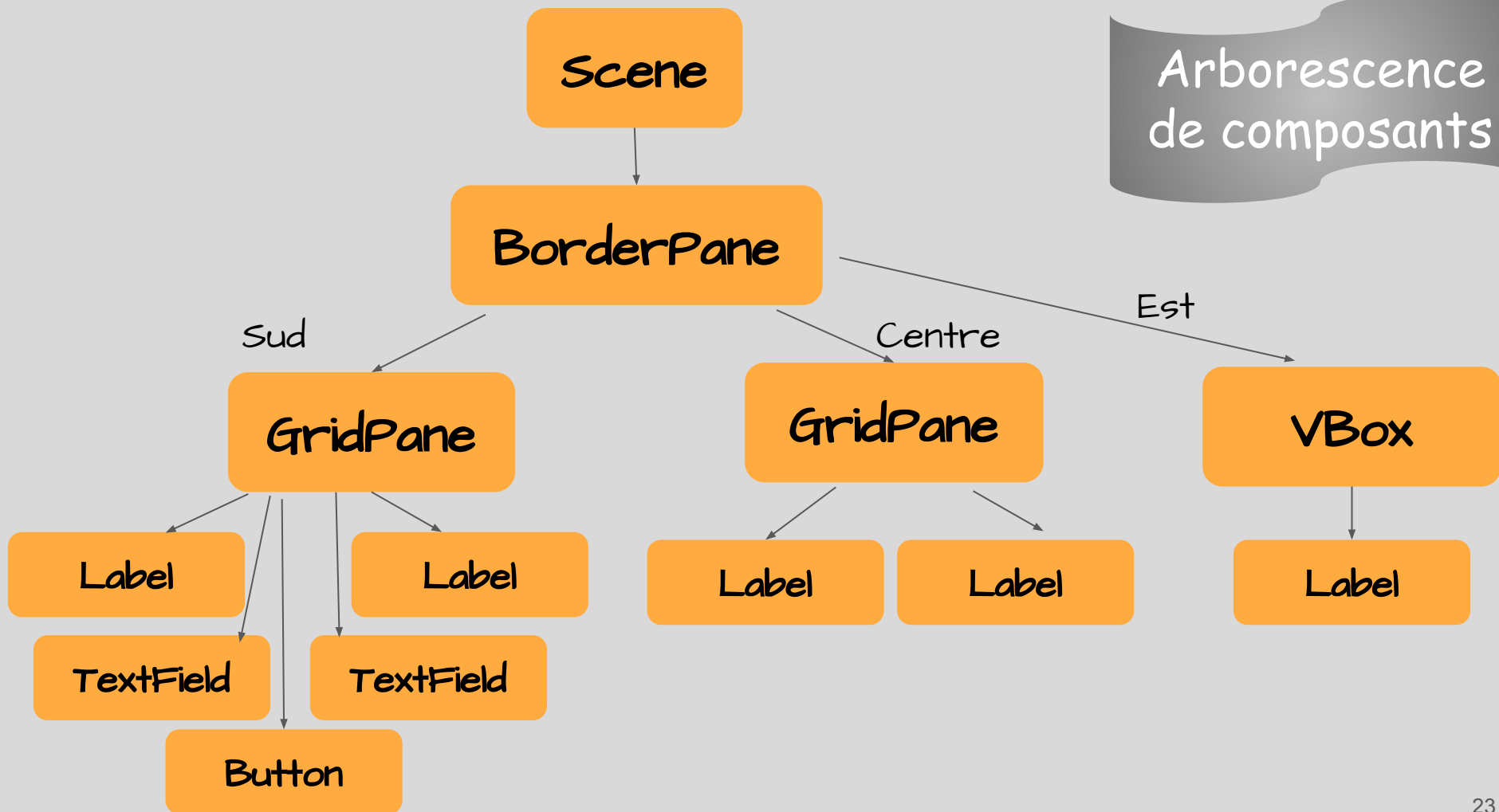
Titre

Compositeur

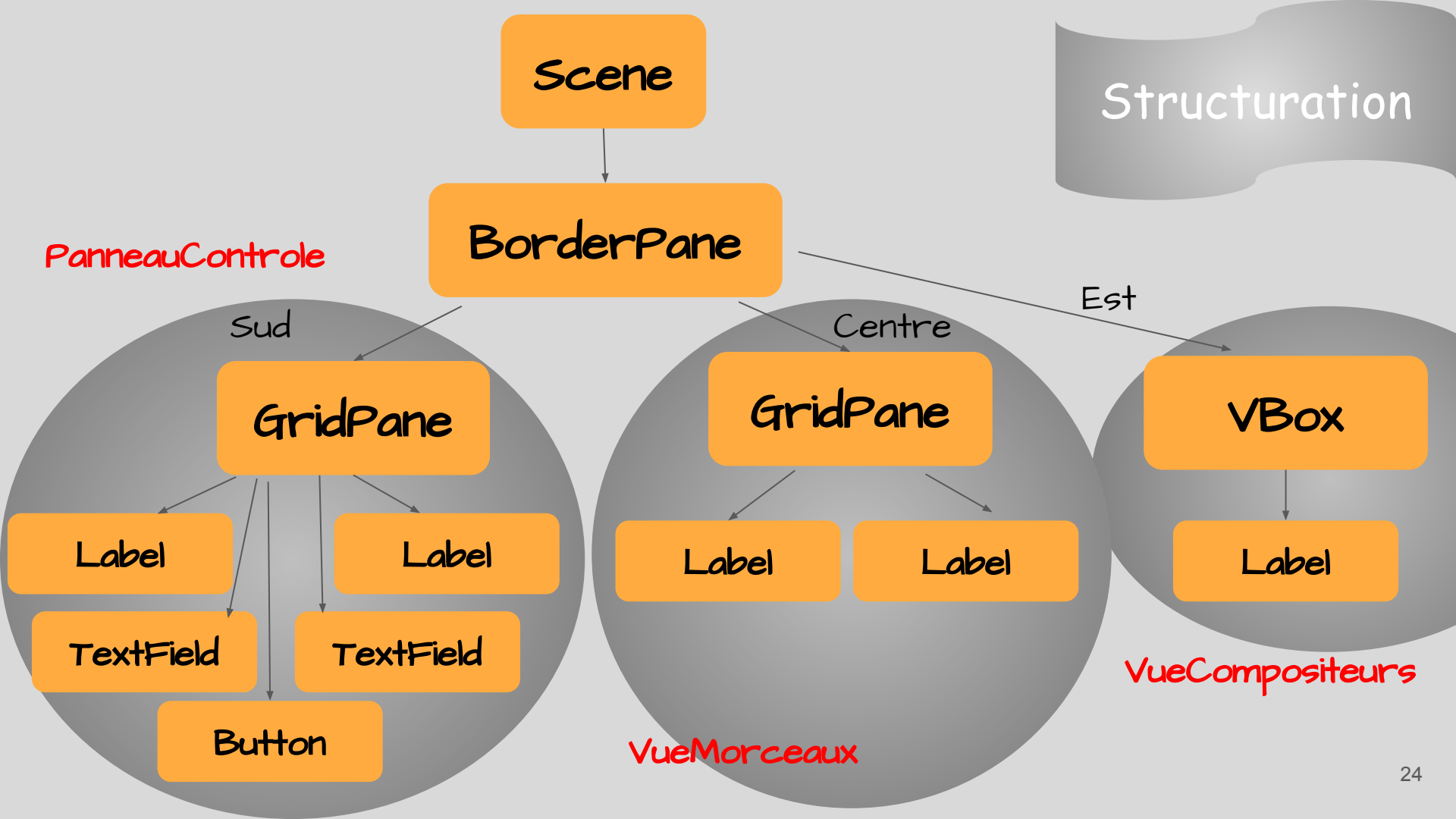
Tous les morceaux ajoutés

Tous les compositeurs

Arborescence de composants



Structuration



▲ Apprendre à

- créer un composant autonome pour chaque vue

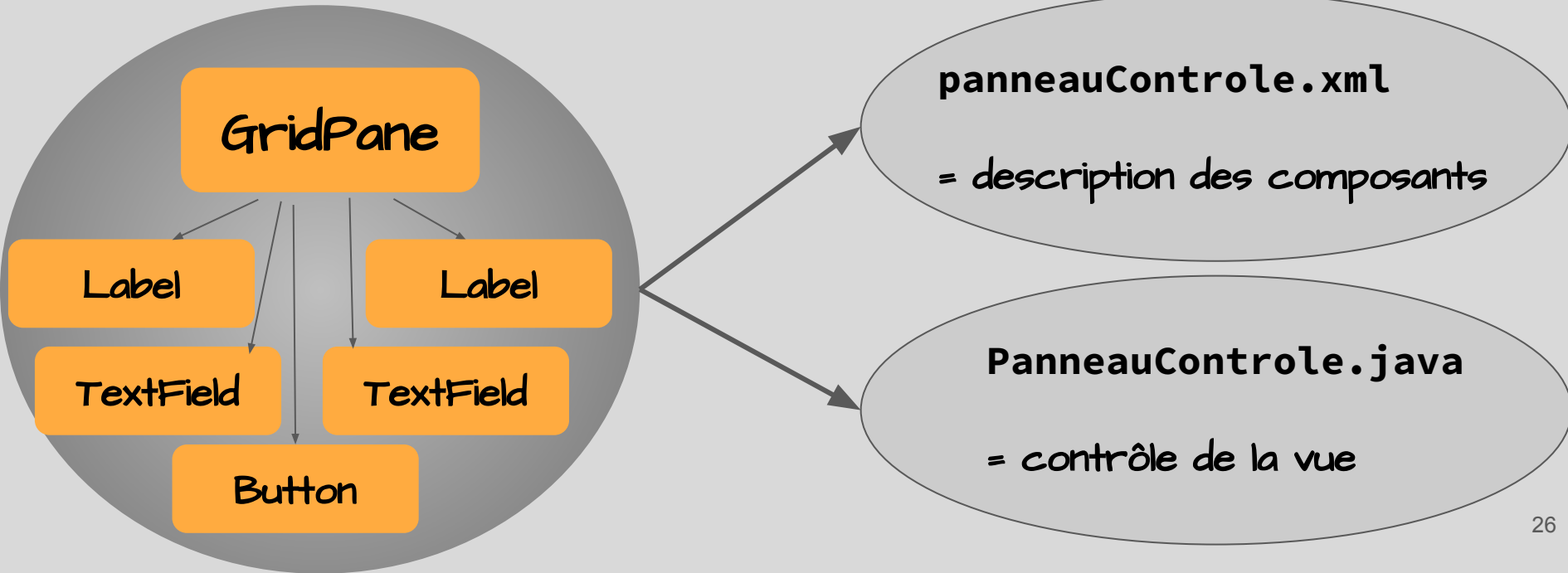
PanneauControle

VueMorceaux

VueCompositeurs

- donner à chaque composant l'accès au modèle
- donner à chaque composant l'accès à ses constituants

PanneauControle



```
<GridPane . . . fx:controller="playList.view.PanneauControle">  
  
    <Button    mnemonicParsing="false"  
              onAction="#ajouterTitre"  
              text="Nouveau" />  
  
</GridPane>
```

```
package playList.view ;  
class PanneauControle {  
  
    public void ajouterTitre() {  
        System.out.println("Il va falloir changer ça ... ");  
    }  
}
```

▲ Apprendre à

- créer un composant autonome pour chaque vue
- donner à chaque composant l'accès au modèle
- donner à chaque composant l'accès à ses constituants

```
Parent root =  
FXMLLoader.load(getClass().getResource("view/panneauControle.fxml"));
```

Contrôleur avec constructeur sans paramètre

```
Playlist playList = new Playlist(); // Le modèle  
  
FXMLLoader loader = new FXMLLoader();  
loader.setLocation(getClass().getResource("view/panneauControle.fxml")  
)  
loader.setControllerFactory(iC->new PanneauControle(playList));  
Parent panneau = loader.load();
```

..... avec paramètre

```
package playList.view ;  
class PanneauControle {  
  
    private PlayList playList ;  
  
    public PanneauControle(PlayList pl) {  
        this.playList = pl ;  
    }  
  
    public void ajouterTitre() {  
        System.out.println("Il va falloir changer ça ... ");  
    }  
}
```

▲ Apprendre à

- créer un composant autonome pour chaque vue
- donner à chaque composant l'accès au modèle
- donner à chaque composant l'accès à ses constituants
 - nommer les constituants en xml
 - injecter les noms dans le contrôleur

```
<GridPane prefHeight="81.0" prefWidth="251.0"
    fx:controller="playList.view.PanneauControle">
    <children>
        <Label text="Titre" GridPane.columnIndex="1"
            GridPane.rowIndex="1" />
        <Label text="Compositeur" GridPane.columnIndex="1"
            GridPane.rowIndex="2" />
        <TextField fx:id="titre" GridPane.columnIndex="2"
            GridPane.rowIndex="1" />
        <TextField fx:id="compositeur" GridPane.columnIndex="2"
            GridPane.rowIndex="2" />
        <Button ... />
    </children>
</GridPane>
```

Avec ou sans
SceneBuilder

Injecter les
noms dans le
contrôleur

```
package playList.view ;
class PanneauControle {

    @FXML
    private TextField titre ;
    @FXML
    private TextField compositeur ;
    private PlayList playList ;
    public PanneauControle(PlayList pl) {
        this.playList = pl ;
    }
    public void ajouterTitre() {
        playList.ajouter(titre.getText(),
                        compositeur.getText());
    }
}
```

▲ Attention à l'ordre d'exécution

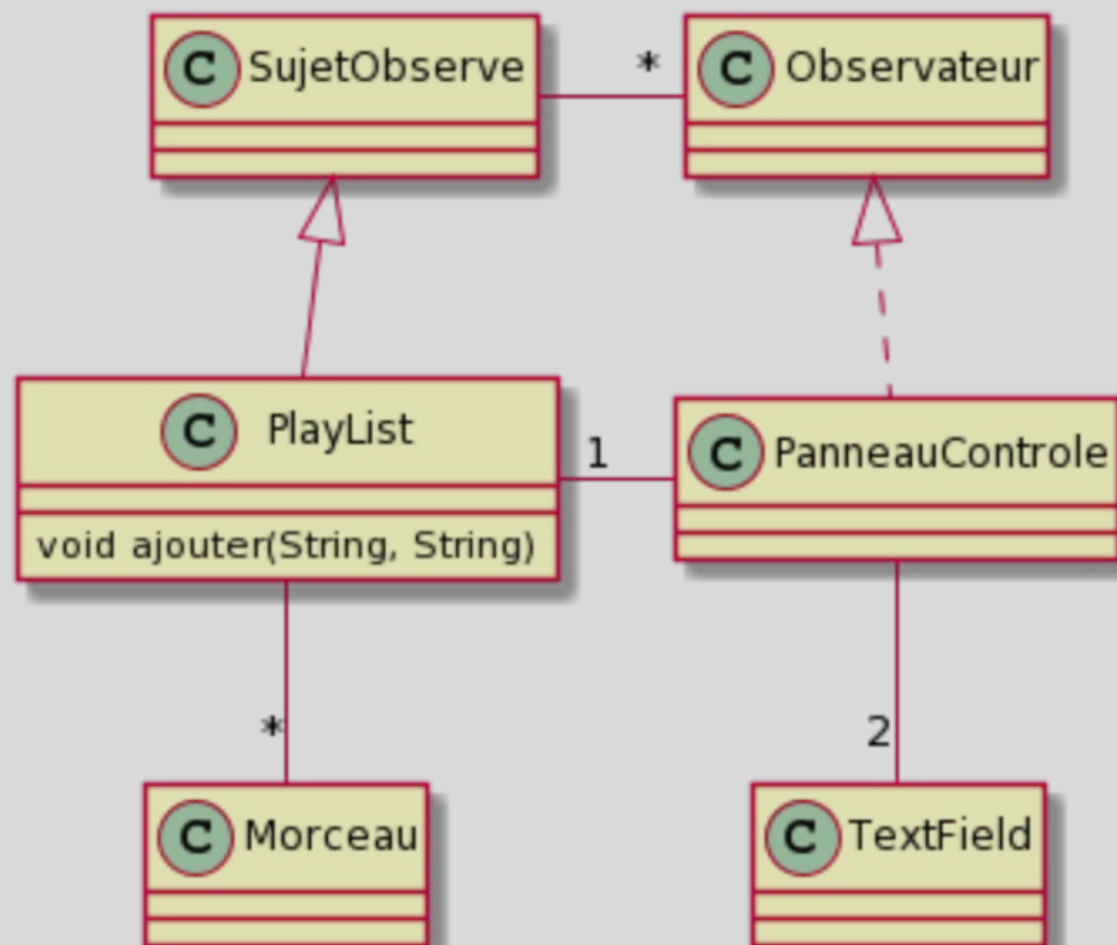
- Appel du constructeur
- Puis injection des composants graphiques

▲ Si besoin, ajouter la fonction

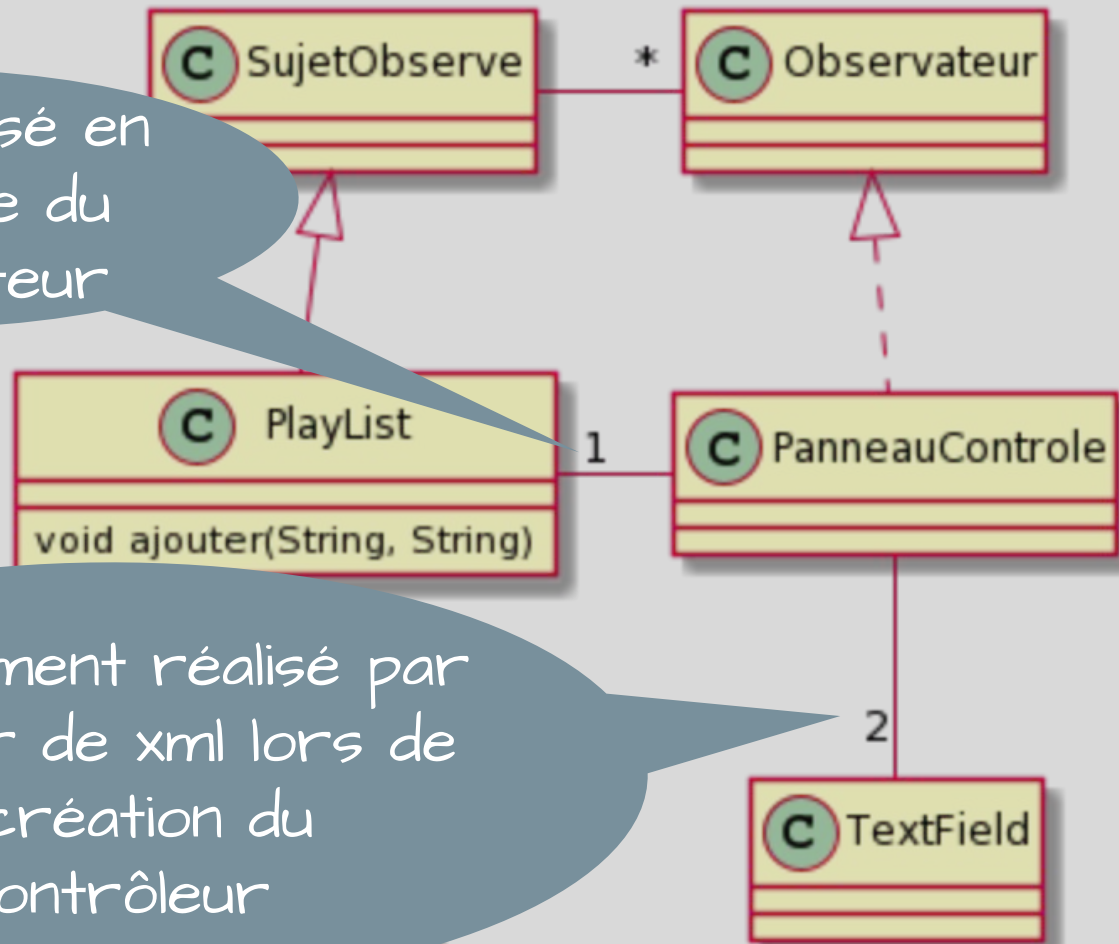
@FXML

void initialize()

Appelée juste après l'injection, avant l'affichage.



Modèle passé en paramètre du constructeur



Attachement réalisé par le loader de xml lors de la création du contrôleur

- ▲ Deux solutions pour écrire la fonction **start**
- Créer et remplir à la main un panneau
 - Un loader par composant rangé dans le panneau
 - Créer un panneau dont la composition est décrite par un fichier XML (clause import)
 - Un seul loader pour tout le panneau

```
PlayList playList = new PlayList(); // Le modèle  
BorderPane panneau = new BorderPane() ;
```

```
FXMLLoader loader = new FXMLLoader();  
loader.setLocation(getClass().getResource("view/panneauControle.fxml"));  
loader.setControllerFactory(iC->new PanneauControle(playList));  
panneau.setBottom(loader.load());
```

```
FXMLLoader loader = new FXMLLoader();  
loader.setLocation(getClass().getResource("view/vueMorceaux.fxml"));  
loader.setControllerFactory(iC->new VueMorceaux(playList));  
panneau.setCenter(loader.load());
```

...

```
primaryStage.setScene(new Scene(panneau, 500, 700));
```

```
<AnchorPane fx:controller="playList.view.Principale">  
...  
  <fx:include fx:id="morceaux" source="morceaux.fxml" />  
  <fx:include fx:id="compositeurs" source="compositeurs.fxml" />  
...  
</AnchorPane>
```

Inclusion de fichiers xml dans la
description de la fenêtre
principale

```
Playlist pl = new Playlist() ;
FXMLLoader loader = new FXMLLoader();
loader.setLocation(getClass().getResource("vues/principale.fxml"));
Principale pc = new Principale(pl);
Morceaux mc = new Morceaux(pl);
Compositeurs cc = new Compositeurs(pl);
...
loader.setControllerFactory(ic -> {
    if (ic.equals(playlist.vues.Principale.class)) return pc;
    else if (ic.equals(playlist.vues.Morceaux.class)) return mc;
    else if (ic.equals(playlist.vues.Compositeurs.class)) return cc;
    else if ...
    else // par défaut...
    return null ;
});
root = loader.load() ;
```