Développement d'applications avec IHM

Dévéloppement en JavaFX

Architecture Logicielle

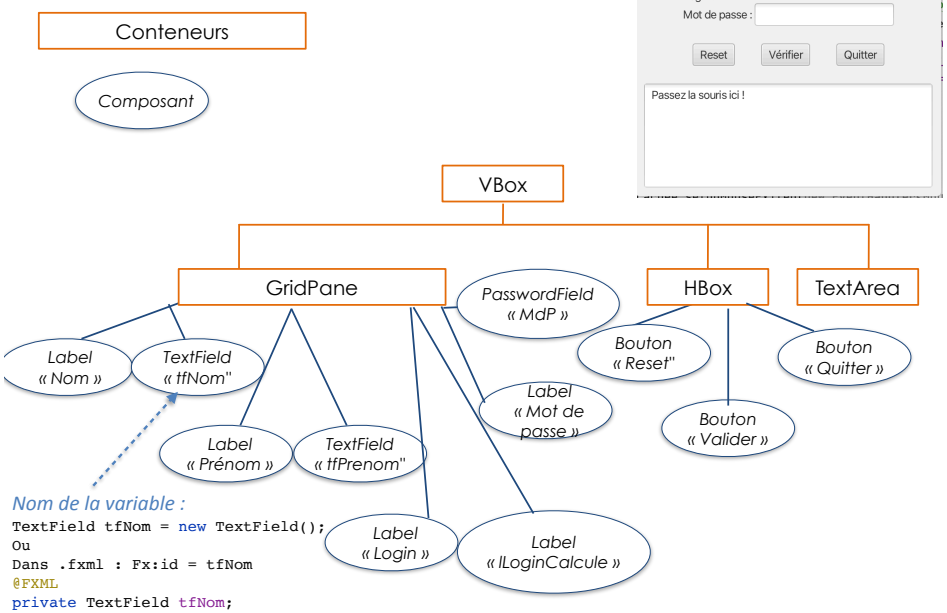
Sophie Dupuy-Chessa
Sophie.Dupuy-Chessa@univ-grenoble-alpes.fr

Rappel - Graphe de scène

Un graphe de scène est un graphe acyclique orienté (arbre orienté) avec :

- une racine qui correspond à un conteneur
- des nœuds (Node) qui sont des éléments de l'interface (conteneurs ou composants graphiques de l'interface)
- des arcs qui représentent les relations parent-enfant

Exemple



Rappel - Noeuds d'une interface

3 types de Node :

- Formes primitives (Shape) 2D ou 3D
- **Conteneurs** (Layout-Pane) qui se chargent de la disposition (layout) des composants enfants et qui ont comme classe parente Pane.
- **Composants**
  - **Composants standard (Controls)**
  - Composants spécialisés qui sont dédiés à un domaine particulier
    - lecteur multimédia, navigateur web, …

JavaFX - démarrer une application

3 classes de base d'une application JavaFX : Application, Stage et Scene

Pour démarrer une application JavaFX :

- La classe principale d'une application javaFX hérite de la classe javafx.application.Application (classe abstraite). Cette classe javafx.application.Application gère tout le cycle de vie de l'application pour vous (ouverture des fenêtres, initialisations, le démarrage et la fin de l'application, etc).

```
public class TP_App extends Application
```

- Après avoir exécuté la méthode Application.launch() – que l'on trouve dans le main() - l'application JavaFX s'initialise et appelle la méthode start() pour démarrer :

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

- C'est la méthode start() le point d'entrée pour toute application JavaFX.

```
public class TP3_Exo1App extends Application {
    @Override
    public void start(Stage stage) – stage est construit par la plateforme
```


JavaFX - Créer des composants

Code Java Pur

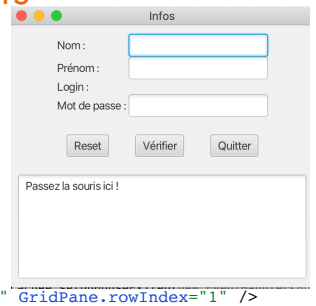
```
VBox vbox = new VBox();
GridPane gridPane = new GridPane();
gridPane.setPadding(new Insets(0, 20, 20, 20));
gridPane.setAlignment(Pos.CENTER);


 ….
TextField tfNom = new TextField();
TextField tfPrenom = new TextField();
Label labelLoginCalcule = new Label();
PasswordField pfMdP = new PasswordField();
 …

gridPane.add(tfNom, 1, 0);
gridPane.add(tfPrenom, 1, 1);
gridPane.add(labelLoginCalcule, 1, 2);
gridPane.add(pfMdP, 1, 3);

 …
Button btnReset = new Button("Reset");
```


JavaFX - Créer des composants



Code Java avec fichier fxml

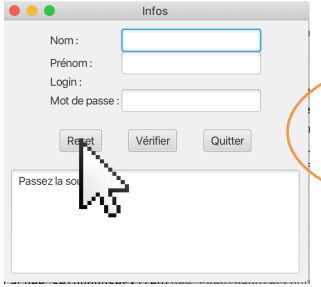
▸ Fichier fxml

```
<VBox …>
   <children>
      <GridPane>
            …
        <children>
           …
           <TextField fx:id="tfNom" … GridPane.columnIndex=« 1" />
            <TextField fx:id="tfPrenom" … GridPane.columnIndex=« 1" GridPane.rowIndex="1" />
            <PasswordField fx:id="pfMdP" GridPane.columnIndex="1" GridPane.rowIndex="3" />
         </children>
      </GridPane>
      <HBox alignment="CENTER" … >
         <children>
            <Button mnemonicParsing="false" … text=« Reset" />
             …
        </children>
      </HBox>
…
```

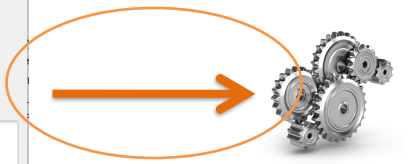
▸ Fichier TP2_Exo2Controller.java (fx:controller)

```
@FXML
private TextField tfNom;
@FXML
private TextField tfPrenom;
…
```


Rappel - Evénement

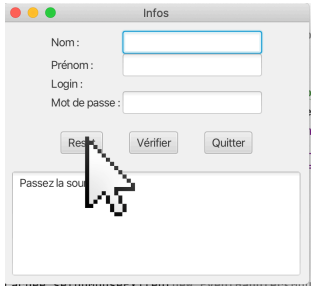


Réception
d'un événement
type "clic de souris"



Rappel : Evénement en JavaFX

- En JavaFX :
  - Un événement correspond à une **structure de données** (la classe javafx.event.Event) qui contient la source de l'événement, la cible et le type de l'événement
  - **La source est l'élément sur lequel l'événement s'est produit et qui l'envoie**
    - Souris, touche, …
  - **La cible est le noeud sur lequel l'événement s'est produit**
    - Bouton, case à cocher,…
  - **Le type d'événement correspond à une classification**
    - Clic souris, touche pressé, déplacement souris (drag),…



=> C'est JavaFX qui reçoit les événements et les transmet grâce à un gestionnaire d'événements (EventHandler).

Exemple - Code Java Pur


```
  ....
TextField tfNom = new TextField();
TextField tfPrenom = new TextField();
Label labelLoginCalcule = new Label();
PasswordField pfMdP = new PasswordField();

  …
Button btnReset = new Button("Reset");

 …


  btnReset.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        tfNom.setText("");
        tfPrenom.setText("");
        labelLoginCalcule.setText("");
        pfMdP.setText("");
    }
});
```

Gestionnaire/Récepteur d'événements ?
Source de l'événement ?
Type d'événement géré ?
Cible de l'événement ?
Méthode de traitement de l'événement ?

Exemple - Code Java + fxml

- Fichier fxml

```
<VBox xmlns="http://javafx.com/javafx/16"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.demo.TP2_Exo2Controller">
  …
<HBox alignment="CENTER" fillHeight="false" prefHeight="50.0" prefWidth="200.0" spacing="30.0">
   <children>
      <Button mnemonicParsing="false" onAction="#resetButtonAction" text="Reset" />
      …
  </children>
</HBox>
```

- Fichier TP2_Exo2Controller.java (fx:controller)

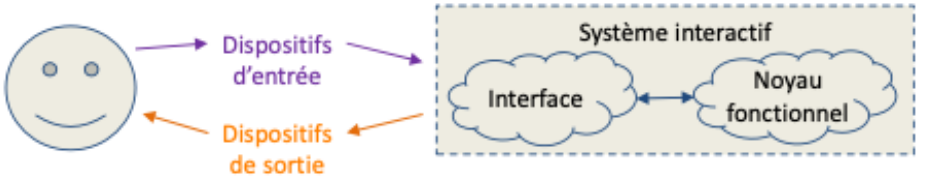
```
@FXML
private TextField tfNom;
@FXML
private TextField tfPrenom;
…
@FXML
private void resetButtonAction(ActionEvent event) {
    tfNom.setText("");
    tfPrenom.setText("");
    labelLoginCalcule.setText("");
    pfMdP.setText("");
}
```

Gestionnaire/Récepteur d'événements ?
Source de l'événement ?
Type d'événement géré ?
Cible de l'événement ?
Méthode de traitement de l'événement ?

Programmation d'une IHM

Modifier / Agir sur l'état du système

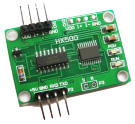
Clavier
Souris
Ecran tactile
Capteur de gestes
Microphone



Percevoir / Comprendre l'état du système

Ecran
Imprimante
Haut-parleur
Vibreur

Programmation d'une IHM



Noyau Fonctionnel

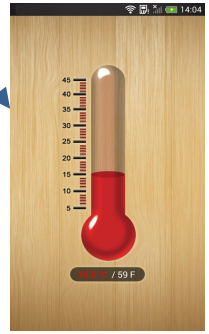
```
float kelvin, celcius;
float fahrenheit;

kelvin = sensor.getValue()

celcius = kelvin - 273.15
fahrenheit = 32 + 1.8 * celsius
```

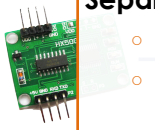
IHM 1

IHM 2



Programmation d'une IHM

IHM 1



Principe élémentaire

Séparer le code

- **Logique applicative (NF)**
- **Interface utilisateur (IHM)**

➡ Séparer
- ➡ les préoccupations
- ➡ Les expertises

IHM 2

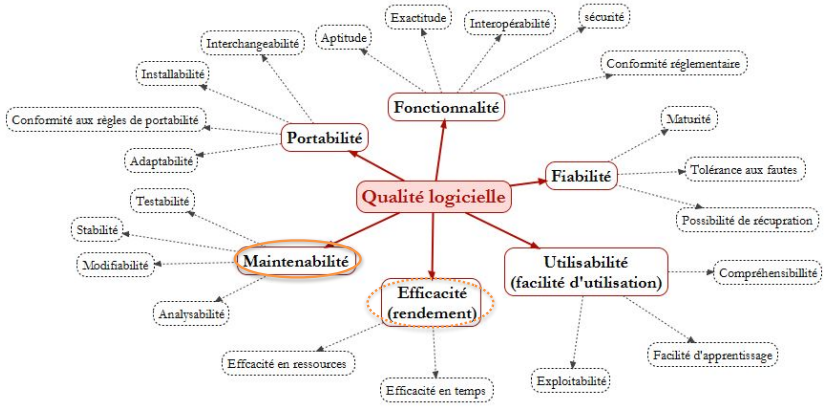
Programmation d'une IHM

Séparation du code

- Noyau Fonctionnel
  - Logique de l'application
  - Modèle, persistance des données
- Interface utilisateur
  - Affichage des éléments graphiques
  - Traitement des entrées utilisateur
  - Réaction aux événements : dialogue
    - Échanges avec le noyau fonctionnel
    - Retours (visuels, auditif, etc)

Architecture logicielle

- **Structuration** du système à développer en un ensemble de composants ayant entre eux des relations bien définies, possédant certaines propriétés requises et respectant certaines contraintes.
  - Objectif : garantir **la qualité du logiciel**
    (cf cours de gestion de projet)



Modularité

- **Compréhension modulaire**
  - chaque module doit pouvoir être réalisé et compris par un programmeur d'une façon relativement indépendante des autres.
- **Continuité modulaire**
  - l'impact d'une modification dans un module doit être réduit à un minimum de modules.
- **Protection modulaire**
  - l'effet d'une erreur se produisant lors de l'exécution d'un module doit rester localisé.
- **Réutilisation**
  - un module doit être conçu afin de favoriser sa réutilisation dans d'autres applications.

➡ Nous : module = classe

Principe de modularité

- **Minimiser le nombre d'interconnexions entre classes** : une classe ne doit communiquer qu'avec un nombre minimum d'autres classes.
  - minimiser les effets de propagation à d'autres classes d'une modification dans le code source ou d'une erreur lors de l'exécution (éviter l'effet cascade)

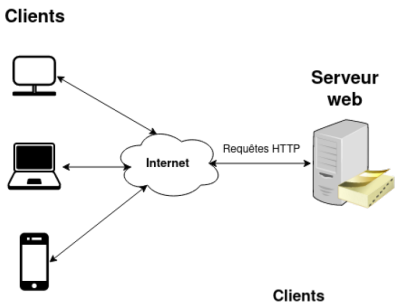
- **Couplage faible** : lorsque 2 classes communiquent entre elles, l'échange d'information doit être minimal (minimiser la taille des interconnexions).
  - éviter variables globales
  - limiter le nombre de paramètres des méthodes publiques

- **Masquage de l'information** : seules les informations qui servent à la communication avec d'autres classes doivent être publiques (visibles de l'extérieur de la classe).

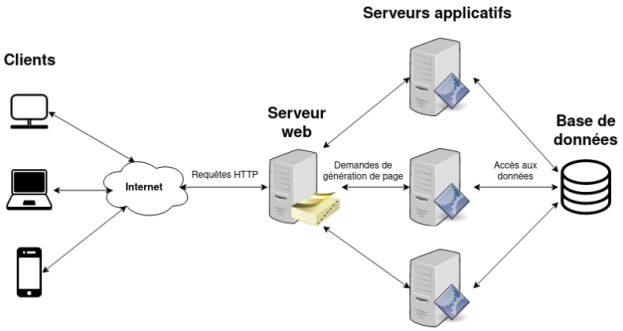
- **Cohérence interne forte** : une classe assure une fonction unique relevant du problème.

- La **taille** d'une classe doit être raisonnable (< 1 000 lignes).

Architecture client-serveur

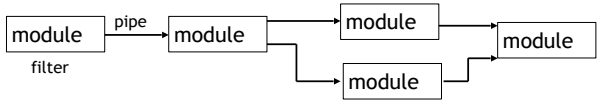


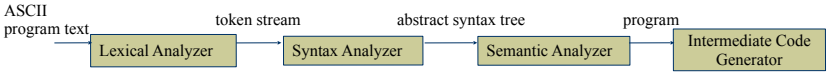
Architecture 3 tiers



Architecture Pipe and Filters

Modules = filtres indépendants reliés par des tuyaux (données à traiter). Transformation incrémentale des données.

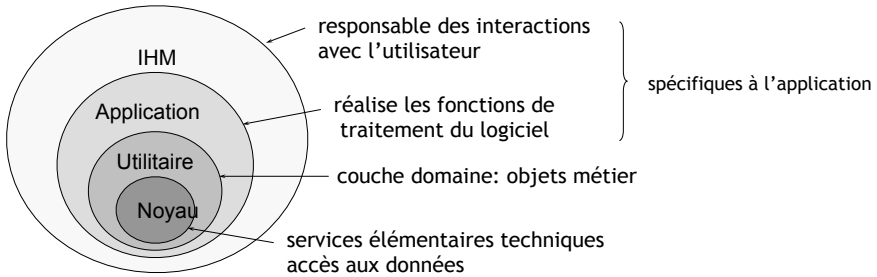




Exemple d'un compilateur

Architecture en couches

- **Motivation** : structuration du logiciel en niveaux d'abstraction
  - chaque niveau réalise une fonction spécifique et est construit sur les niveaux inférieurs
  - plus les couches sont hautes, plus elles sont spécifiques à l'application



Exemple d'architecture en couches pour une application interactive

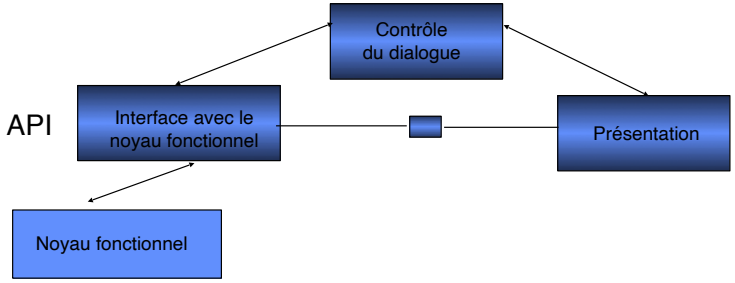
Avantages et inconvénients d'une architecture en couches

- **Avantages**
  - **Séparation des préoccupations**
    - on ne mélange pas les services de bas niveau avec la logique de l'application.
  - **Réduction de la complexité**
    - permet de réduire les détails que le développeur a besoin de connaître à chaque niveau (abstraction).
    - leur implémentation est déférée à des niveaux plus bas (dissimulation).
  - **Maintenance**
    - une modification d'une couche n'affecte pas les couches de niveau inférieur.
    - une modification de l'**implémentation** d'une couche n'affecte pas les couches de niveau supérieur.
  - **Réutilisation**
    - les couches les plus basses peuvent être communes à plusieurs applications.

- **Inconvénients**
  - Performance : il faut traverser toutes les couches pour parvenir aux fonctions de bas niveaux.
  - La modularité est difficilement respectée.
  - Les dépendances entre noyau fonctionnel et interface rendent difficiles les modifications.
    - ➡ **Besoin d'un élément entre le noyau fonctionnel et l'interface**

Architecture de systèmes interactifs : Seeheim (1983)

- *La présentation* est la couche en contact direct avec les entrées et sorties. Elle interprète les actions de l'utilisateur (dispositifs physiques) et génère les sorties (affichage) au niveau lexical.
- *Le contrôleur de dialogue* gère le séquencement de l'interaction en entrée et en sortie. Il maintient un état lui permettant de gérer les modes d'interaction et les enchaînements d'écrans.
- *L'interface du noyau fonctionnel* est la couche adaptative entre le système interactif et le noyau fonctionnel. Elle convertit les entrées en appels du noyau fonctionnel et les données abstraites de l'application en des objets présentables à l'utilisateur.

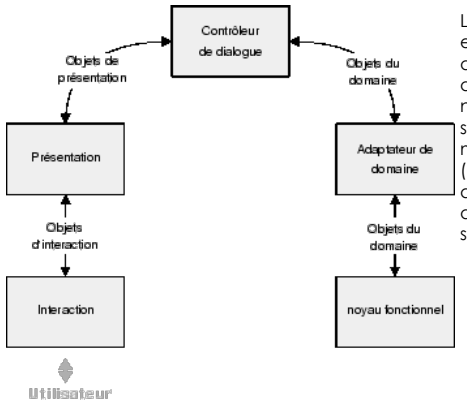


Architecture de systèmes interactifs : Modèle ARCH (1992)

Le contrôleur de dialogue joue le même rôle articulatoire que celui du modèle de Seeheim. Il est notamment responsable du séquencement des tâches et du maintien de la consistance entre les vues multiples.

Le composant de présentation joue le rôle de médiateur entre le composant d'interaction et le contrôleur de dialogue. Il maintient une représentation logique des widgets qui est indépendante de la plate-forme.

Le composant d'interaction désigne l'ensemble des widgets (objets interactifs) d'une boîte à outils, ainsi que les communications avec les périphériques physiques. Il dépend d'une plate-forme donnée.



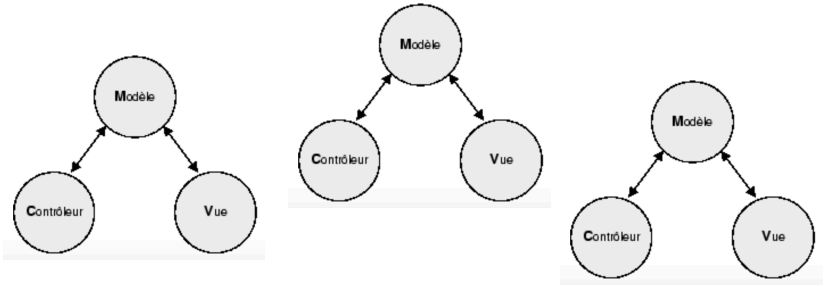
L'*adaptateur du domaine* est responsable des tâches dépendantes du domaine qui ne font pas partie du noyau fonctionnel mais qui sont nécessaires à sa manipulation par l'utilisateur (réorganisation des données du domaine, détection des erreurs sémantiques)

Le noyau fonctionnel représente la partie non interactive de l'application.

Modèle de référence MVC

A l'origine, le modèle MVC est un modèle à agents ou chaque agent est constitué de :

- *Le modèle* est le noyau fonctionnel de l'agent. Il peut représenter des données brutes (entier, chaîne de caractères) ou des objets ayant un comportement complexe. Il notifie les vues qui lui sont associées à chaque fois que son état se trouve modifié par le noyau de l'application ou par ses contrôleurs.
- *La vue* maintient une représentation du modèle perceptible par l'utilisateur, qu'elle met à jour à chaque changement d'état du modèle. Elle est en général constituée d'objets graphiques.
- *Le contrôleur* reçoit et interprète les événements utilisateur, en les répercutant sur le modèle (modification de son état) ou sur la vue (retour instantané).



Modèle MVC

Modèle : définit comment les données peuvent être créées, récupérées, stockées, modifiées.

Type de questions relatives :

Quel format doit prendre l'adresse postale ?
Comment gérer deux utilisateurs qui s'inscrivent et veulent prennent simultanément la dernière place ?
Un utilisateur doit-il pouvoir mettre à jour son adresse électronique, et comment doit-elle être validée ?
➡Fournit les méthodes que le contrôleur va pouvoir appeler pour modifier les données

Vue : représentation visuelle des données et support aux actions utilisateur pour la réalisation de sa tâche.

(Elle peut être constituée de plusieurs fenêtres qui s'enchainent)

Type de questions relatives :

Que se passe-t-il lorsqu'un utilisateur clique sur le bouton "OK" ?
Combien d'enregistrements dois-je afficher dans mon tableau à la fois ?
Quels champs doivent être remplis avant que le bouton OK ne soit activé ?
➡Faire le lien avec le modèle afin que les modifications apportées au modèle soient instantanément répercutées sur l'utilisateur.
➡Invoque des méthodes dans le contrôleur qui reflètent les interactions de l'utilisateur avec la vue.

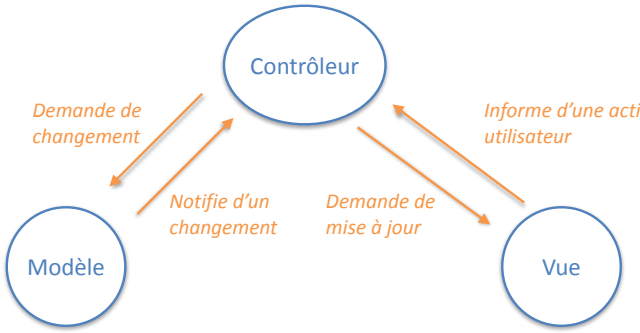
Contrôleur : gère le lien entre la vue et le modèle

Type de questions relatives :

Demander une modification de l'adresse au modèle quand l'utlisateur clique sur « OK » ?
➡Fournit des méthodes que la vue peut invoquer en fonction des interactions de l'utilisateur.
➡Invoque des méthodes dans le modèle pour le mettre à jour en fonction des interactions de l'utilisateur ou des processus d'arrière-plan du contrôleur.

Modèle MVC

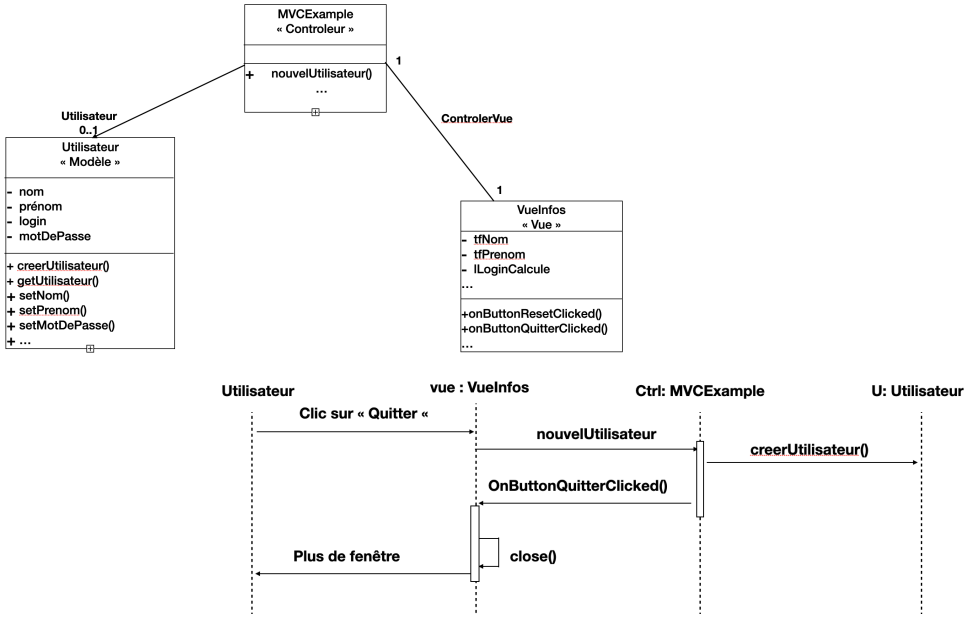
De nombreuses variantes existent !



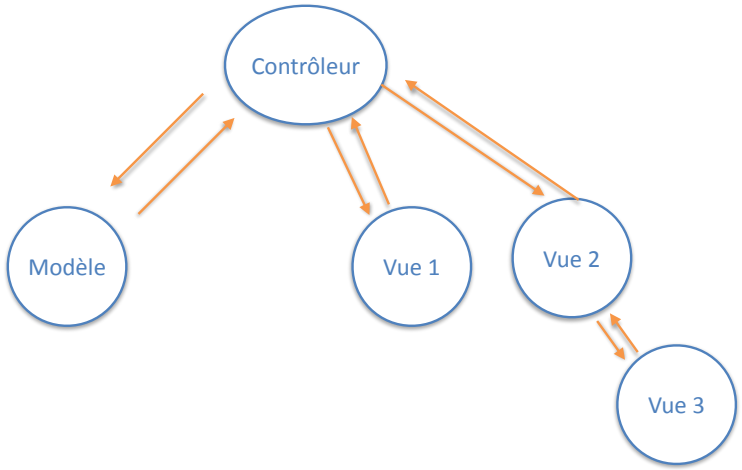
Remarques :

- Une vue ne représente pas forcément toute une fenêtre.
- Un modèle peut être visualisé au travers de plusieurs vues.
- Les vues ne montrent pas forcément tout le modèle.

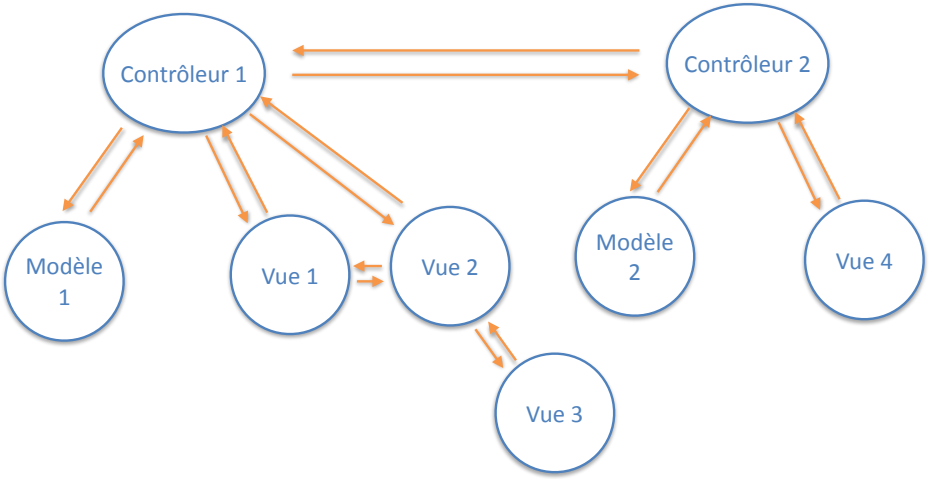
Modèle MVC - Exemple simplifié



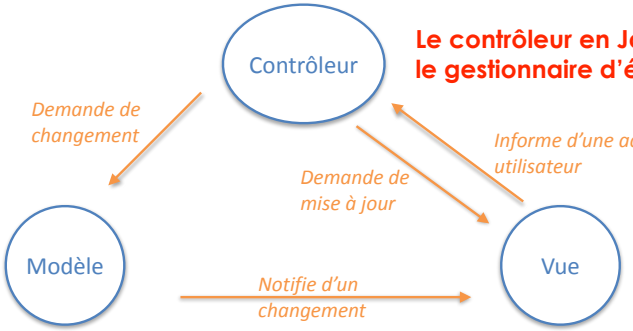
Modèle MVC



Modèle MVC



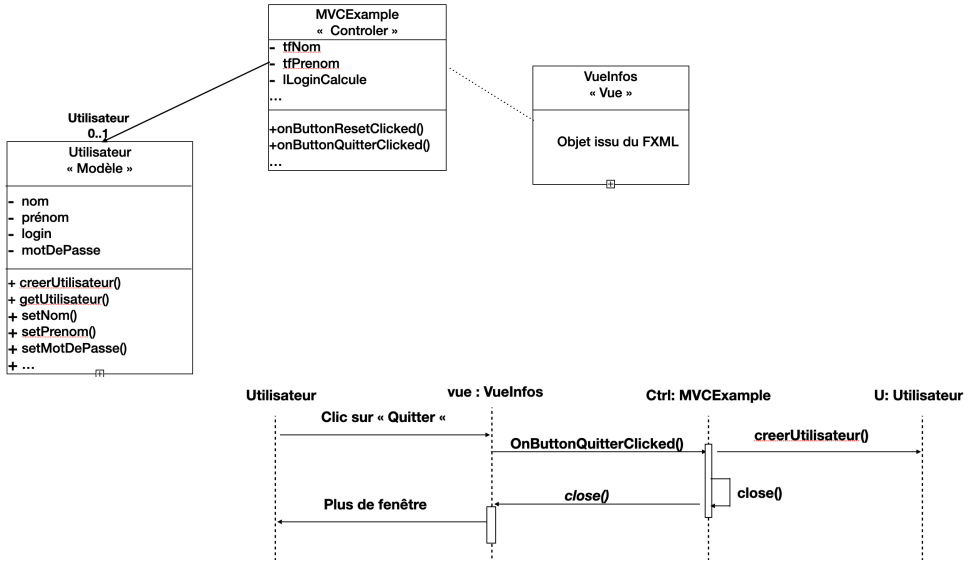
Modèle MVC - variante JavaFX



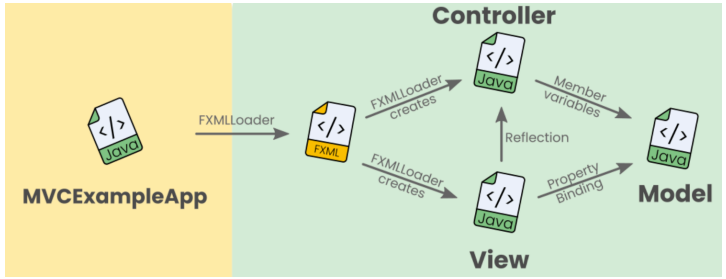
Le contrôleur en JavaFX est aussi le gestionnaire d'événements

Les composants de la vue sont contenus dans le fichier FXML et gérés par le contrôleur.

MVC en JavaFX - Exemple simplifié



Modèle MVC - variante JavaFX



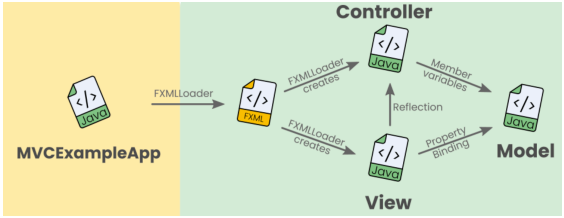

```
public class HelloApplication extends Application {

    @Override
    public void start(Stage stage) throws IOException {
        // déclaration du contrôleur contenu dans hello-view.fxml
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-
view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 500, 320);
        …
        stage.setScene(scene);
        stage.show();

    }

  …   }
```


Modèle MVC - variante JavaFX




```
public class MVCExample{

public Utilisateur util;

@FXML
private TextField fieldNom, fieldPrenom;

…

@FXML
public void onButtonQuittered(ActionEvent event)  throws Exception {
    try {
        // objet qui va charger une hiérarchie d'objets à partir du fichier xml pour l'affichage
    FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("affichage.fxml"));

    // Associer le controleur courant v1
    fxmlLoader.setController(this);
…
```


Architecture pour la SAE