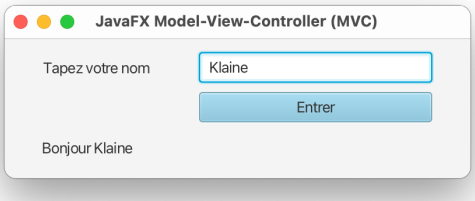
TP4 – Implémentation des patrons d'Architecture (MVC-MVP-MVVM)

Avant-Propos

L'objectif **de tous les exercices** sera d'implémenter en **Java** à l'aide du **Framework JavaFX** une fenêtre de connexion GUI dans laquelle l'utilisateur rentre son nom dans le champ texte puis presse sur le bouton [**Entrer**]. L'application GUI affichera en guise de retour un message de bienvenue personnalisé (qui s'adressera à un inconnu si aucun nom n'a été donné).



A l'instar du TP3 lors de l'exercice d'introduction au Framework d'introduction à JavaFX, le développement de l'application GUI s'appuiera toujours sur les fichiers :

- **myscreen.fxml** (contenant l'interface de l'application GUI) ;
- **MyView.java** (contenant la classe de contrôle) ;
- **MyApplication.java** (contenant la classe d'entrée de l'application).

Vous noterez tout d'abord que si l'architecture sera à chaque fois différente, le contenu de la fenêtre décrit dans le fichier **myscreen.fxml** sera à chaque fois le même :

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane prefHeight="117.0" prefWidth="392.0" xmlns:fx=http://javafx.com/fxml
   fx:controller="MyView">
   <children>
      <Label layoutX="33.0" layoutY="17.0" text="Tapez votre nom" />
```

```
     <TextField layoutX="167" layoutY="12" prefWidth="200" fx:id="m_input" />
     <Button defaultButton="true" layoutX="167.0" layoutY="46.0" prefWidth="200.0"
text="Entrer" onAction="#handleSayHello" />
     <Label id="console" layoutX="32.0" layoutY="81.0" prefHeight="26.0"
prefWidth="335.0" fx:id="m_output" />
   </children>
</AnchorPane>
```

En particulier, vous pouvez identifier en analysant le contenu du fichier les composants graphiques interactifs de l'application :

- la classe **MyView** qui recevra les événements d'interaction ;
- la méthode **MyView.handleSayHello()** sera exécutée en réponse à l'événement d'appui sur le bouton [**Entrer**] ;
- le champ de texte **MyView.m_input** permettra à l'utilisateur de saisir son nom ;
- le label **MyView.m_output** réceptionnera le message de bienvenue.

Vous noterez également que c'est la classe qui encapsule la vue **MyView** qui joue le rôle de **classe de contrôle en JavaFX**.

Le terme est donc trompeur, non seulement la **classe de contrôle en JavaFX** n'a donc rien à voir avec le **contrôleur du modèle MVC**, mais en plus il s'agit en fait de la **vue dans les modèles MVC, MVP et MVVM** !

Exercice 1 : Modèle-Vue-Contrôleur

Préambule :

Voici l'architecture **Modèle-Vue-Contrôleur** (**MVC**) de la fenêtre de connexion :

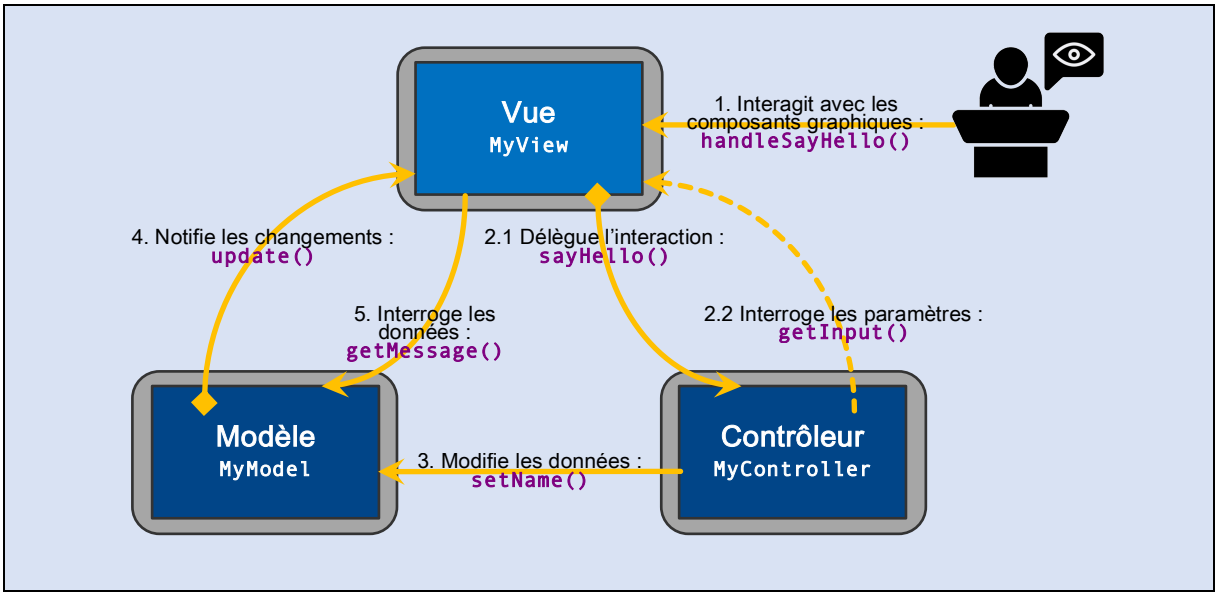


Schéma de l'**architecture MVC** (avec les classes et méthodes associées)

En particulier, vous n'utiliserez qu'une seule vue et qu'une seule donnée (le nom de l'utilisateur) qui sera stockée indépendamment de l'interface.

L'analyse de l'architecture montre le schéma du parcours des informations :

1. l'**utilisateur** entre son nom puis valide en appuyant sur [**Entrer**] dans l'écran de connexion : cela déclenche la méthode **handleSayHello()** de la **vue** ;
2. la **vue** délègue alors l'interaction au **contrôleur** : elle appelle la méthode **sayHello()** du **contrôleur** ;
   le **contrôleur** interroge la **vue** pour récupérer le nom de l'utilisateur : il appelle la méthode **getInput()** de la **vue** ;
3. le **contrôleur** modifie le nom stocké dans le **modèle** : il appelle la méthode **setName()** du **modèle** ;
4. le **modèle** notifie les changements à la **vue** : il appelle la méthode **update()** de la **vue** ;
5. la **vue** interroge le **modèle** pour récupérer le message : il appelle la méthode **getMessage()** du **modèle**. Dans la suite, la génération du message de bienvenue sera confiée à un **service** et sa méthode **sayHello()**.

Exercice :

Le but de l'exercice est donc d'implémenter l'application GUI de connexion conformément à l'**architecture MVC**.

Grâce au parcours des informations, vous allez donc compléter les fichiers :

- **MyApplication.java** (contenant la classe d'entrée)
- **MyView.java** (décrivant la vue),
- **MyModel.java** (décrivant le modèle)
- **MyController.java** (décrivant le contrôleur)

qui sont dans le dossier **/workspace/11-LocalMVC/src** du **conteneur Docker**.

On fournit également un fichier **MyService.java** qui encapsule la classe de service de délivrance du message de bienvenue qui sera envoyé en guise de réponse :

Pour compiler et exécuter votre code, dans le **terminal Docker**, lancez :

- `run.sh 11-LocalMVC`

Exercice 2 : Modèle-Vue-Présentation

Introduction :

L'organisation de l'application devra se conformer à l'architecture **Modèle-Vue-Présentation** (**MVP**).

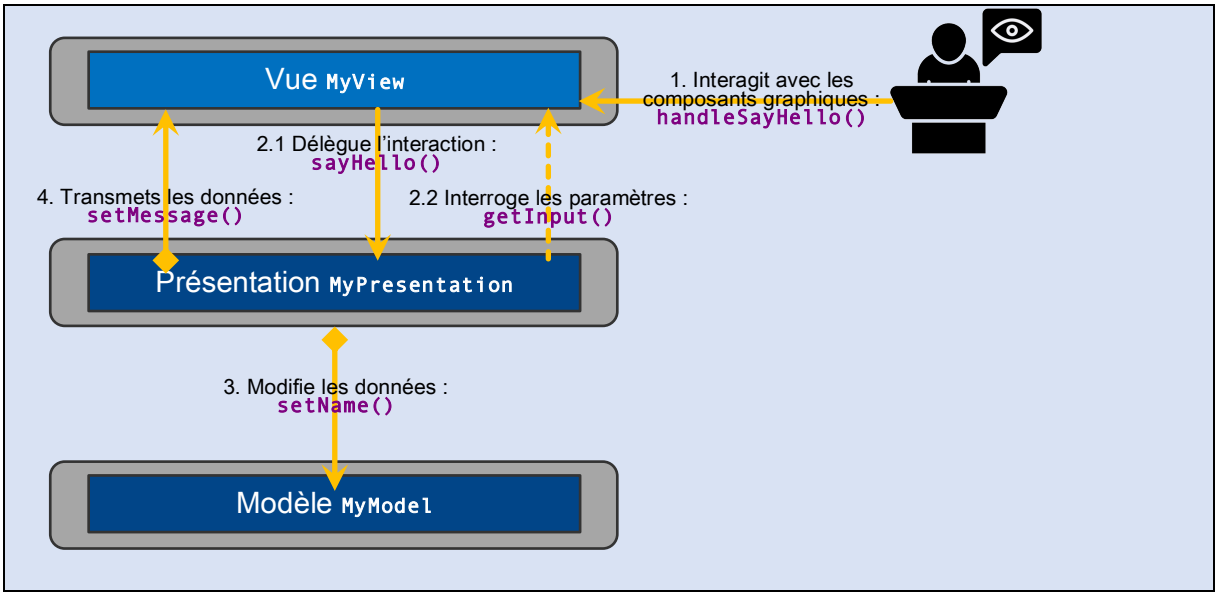


Schéma de l'**architecture MVP** (avec les classes et méthodes associées)

En particulier, vous n'utiliserez qu'une seule vue et qu'une seule donnée (le nom de l'utilisateur) qui sera stockée indépendamment de l'interface.

L'analyse de l'architecture montre le schéma du parcours des informations :

1. l'**utilisateur** entre son nom puis valide en appuyant sur [**Entrer**] dans l'écran de connexion : cela déclenche la méthode **handleSayHello()** de la **vue** ;
2. la **vue** délègue alors l'interaction à la **présentation** : elle appelle la méthode **sayHello()** de la **présentation** ;

la **présentation** interroge la **vue** pour récupérer le nom de l'utilisateur : il appelle la méthode **getInput()** de la **vue** ;

3. la **présentation** modifie le nom dans le **modèle** : elle appelle la méthode **setName()** du **modèle** ;
4. la **présentation** transmets le message à la **vue** : il appelle la méthode **setMessage()** de la **vue**.

Exercice :

Le but de l'exercice est donc d'implémenter l'application GUI de connexion conformément à l'**architecture MVP**.

Grâce au parcours des informations, vous allez donc compléter les fichiers :

- **MyApplication.java** (contenant la classe d'entrée)
- **MyView.java** (décrivant la vue),
- **MyModel.java** (décrivant le modèle)
- **MyPresentation.java** (décrivant la présentation)

qui sont dans le dossier **/workspace/12-LocalMVP/src** du **conteneur Docker**.

Pour compiler et exécuter votre code, dans le **terminal Docker**, lancez :

- `run.sh 12-LocalMVP`

Exercice 3 : Modèle-Vue-Vue/Modèle

Introduction :

L'organisation de l'application devra se conformer à l'architecture **Modèle-Vue-Vue/Modèle** (**MVVM**).

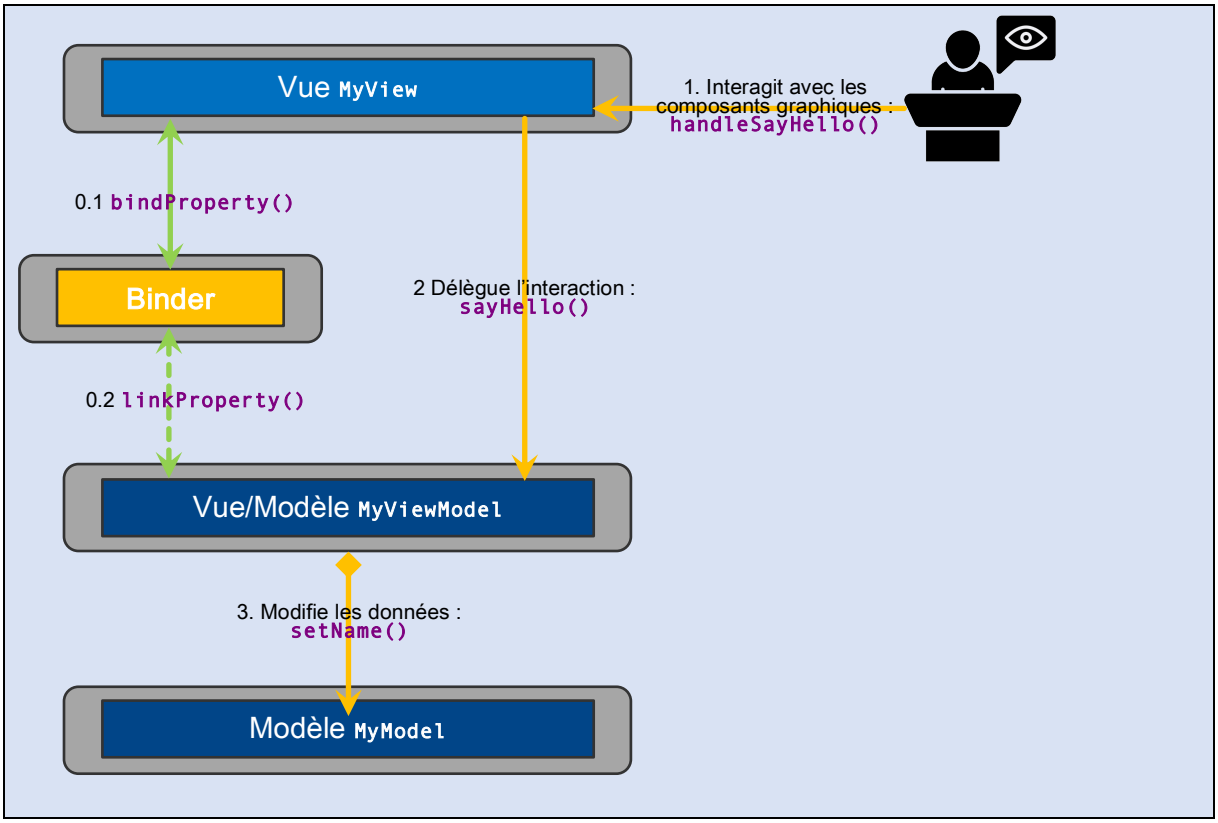


Schéma de l'**architecture MVVM** (avec les classes et méthodes associées)

En particulier, vous n'utiliserez qu'une seule vue et qu'une seule donnée (le nom de l'utilisateur) qui sera stockée indépendamment de l'interface.

Le schéma de l'**architecture MVVM** est très proche de celui de l'**architecture MVP**. La différence c'est que cette fois, la **vue/modèle** intègre deux propriétés :

- la propriété **MyViewModel.m_input** qui est liée au label **MyView.m_input** ;
- la propriété **MyViewModel.m_message** qui est liée au texte **MyView.m_output**.

Le **lien bilatéral** entre les **propriétés de la vue/modèle** et les **propriétés des éléments graphiques de la vue** sont assurées par le **binder** : **toute modification de la vue affecte la vue/modèle** et **réciproquement**.

Les propriétés sont protégées dans des accesseurs en lecture/écriture sécurisés **inputProperty()** et **messageProperty()**.

L'analyse de l'architecture montre le schéma du parcours des informations :

1. l'**utilisateur** entre son nom puis valide en appuyant sur [**Entrer**] dans l'écran de connexion : cela déclenche la méthode **handleSayHello()** de la **vue** ;
2. la **vue** délègue alors l'interaction à la **vue/modèle** : elle appelle la méthode **sayHello()** de la **vue/modèle** qui récupère le **nom de la vue** directement grâce à sa propriété accédée par **inputProperty()** ;
3. la **vue/modèle** modifie le **nom dans le modèle** en appelant la méthode **setName()** du **modèle** ;
4. la **vue/modèle** modifie la **sortie dans la vue** directement en appelant sa propre méthode **setMessage()** qui affecte un texte à la propriété accédée par **messageProperty()**.

Exercice :

Le but de l'exercice est donc d'implémenter l'application GUI de connexion conformément à l'**architecture MVVM**.

Grâce au parcours des informations, vous allez donc compléter les fichiers :

- **MyApplication.java** (contenant la classe d'entrée)
- **MyView.java** (décrivant la vue),
- **MyModel.java** (décrivant le modèle)
- **MyViewModel.java** (décrivant la vue/modèle)

qui sont dans le dossier **/workspace/13-LocalMVVM/src** du **conteneur Docker**.

Les **propriétés des éléments graphiques de la vue** sont accédées par la méthode **textProperty()** que ce soit pour l'entrée de type **Label** ou la sortie de type **Text**. Les **propriétés de la vue/modèle**, de type **TextProperty**, sont créées au moyen du constructeur de **SimpleStringProperty()** que ce soit pour l'entrée ou pour le message.

Les **propriétés de la vue** et de la **vue/modèle** sont liées dans le **binder** par la méthode **bindBidirectional()**. Les chaines de caractère associées aux propriétés de type **TextProperty** sont lues par la méthode **get()** et modifiées par la méthode **set()**.

Pour compiler et exécuter votre code, dans le **terminal Docker**, lancez :

- `run.sh 13-LocalMVVM`