**DAA - Développement d'applications Android**

# Live Data et MVVM

*28 October 2025*

## Table des matières

# 1 LiveData

LiveData est une classe observable qui permet de stocker des données de manière réactive. Elle est conçue pour être utilisée avec le cycle de vie des composants Android, ce qui signifie qu'elle respecte automatiquement le cycle de vie des activités et des fragments.

## 1.1 Description

### Avantages

- L'interface graphique se mets à jour automatiquement lorsque les données changent.
- Évite les fuites de mémoire en respectant le cycle de vie des composants.
- L'observateur est appelé dans le UI-thread –> on peut modifier l'interface graphique directement.

Généralement, les LiveData sont crée dans un ViewModel et permettent de:

- Remplacer la sauvegarde de l'état lors de la re-création d'une activité.
- Partager des données et propager des événements entre plusieurs composants (fragments, activités).

> ⚠ **Warning**
>
> Les `LiveData` ne sont pas modifiables directement. Pour cela, il faut utiliser `MutableLiveData`.

## 1.2 Exemple d'utilisation

### Création d'une LiveData

```
val data = MutableLiveData(0)
```

### Accès à une valeur

```
data.value
```

### Mise à jour d'une valeur

```
data.value = 42     // Synchrone, doit être appelé dans le UI-thread
// ou
data.postValue(42)  // Asynchrone, peut être appelé depuis un thread en arrière-plan
```

## 1.3 Observation des données

Pour observer les changements de données dans une `LiveData`, on utilise la méthode `observe`, qui prend un `LifecycleOwner` (comme une activité ou un fragment) et un observateur (une fonction lambda).

### Depuis une activité

```
data.observe(this) { value ->
    textview.text = "$value" // Mettre à jour l'interface utilisateur avec la nouvelle valeur
}
```

### Depuis un fragment

```
data.observe(viewLifecycleOwner) { value ->
    textview.text = "$value" // Mettre à jour l'interface utilisateur avec la nouvelle valeur
}
```

## 1.4 Particularités

- Le type générique peut-être inféré à partir de la valeur initiale. `val data = MutableLiveData("Toto") --> String`

- En cas d'une initialisation sans valeur, le type doit être spécifié. `val data = MutableLiveData<Int>()`
- Initialisation paresseuse possible avec `by lazy`.
  - `val data : MutableLiveData<String> by lazy { MutableLiveData("Toto") }`
- Les `LiveData` peuvent encapsuler des types complexes, comme des listes ou des objets personnalisés.
  - `val data = MutableLiveData<List<String>>(listOf())`

# 2 MVVM

## 2.1 MVC

Le modèle MVC (Model-View-Controller) divise une application en trois composants principaux:

- Modèle (Model): Gère les données et la logique métier.
- Vue (View): Affiche les données à l'utilisateur et gère l'interface utilisateur.
- Contrôleur (Controller): Interagit avec le modèle et la vue pour traiter les entrées utilisateur et mettre à jour l'interface.

Avec Android, il n'est pas possible d'appliquer strictement le modèle MVC car les widgets ne peuvent pas directement interagir avec le modèle.

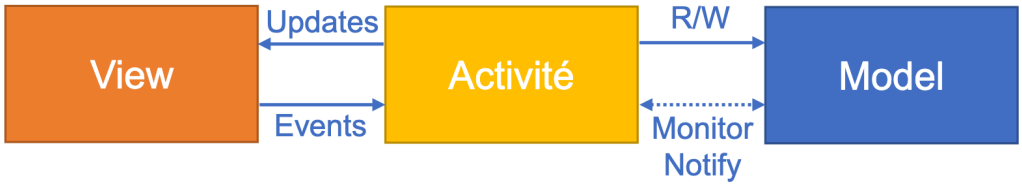


*Fig. 1. – Capture des slides du cours – Architecture MVC*

## 2.2 MVVM