Université Abderrahmane Mira de Béjaia **février 2026**
Faculté des Sciences Exactes
Département d'Informatique
ING4
Architectures Logicielles

Série de TP N° 1

Activité 1

Le patron d'architecture MVP

- Le modèle-vue-présentation (MVP, Model-View-Presenter) est un patron d'architecture, considéré comme un dérivé du patron d'architecture modèle-vue-contrôleur (MVC).
- Il garde les mêmes principes que MVC sauf qu'il élimine l'interaction entre la vue et le modèle parce qu'elle sera effectuée par le biais de la présentation, qui organise les données à afficher dans la vue.
- Il sépare clairement la logique métier de l'interface graphique afin d'améliorer la maintenabilité et la testabilité.
- **Model**
  - Contient les données et la logique métier
  - Indépendant de l'interface utilisateur
- **View**
  - Interface graphique (UI)
  - Affiche les données
  - Transmet les actions utilisateur au Presenter
  - Ne contient aucune logique métier
- **Presenter**
  - Intermédiaire entre Model et View
  - Contient la logique de présentation
  - Met à jour la View en fonction des données du Model

Implémentation du pattern MVP en Java : Projet Gestion de tâches en MVP

- Vous devez développer une application simple de gestion de tâches (To-Do List) en respectant strictement le pattern MVP.
- Fonctionnalités : ajouter, afficher, terminer, supprimer une tâche et quitter l'application.
- L'application fonctionnera en mode console
- L'utilisateur interagit uniquement avec la View.

Activité 2

Le patron d'architecture MVVM

- Le MVVM (Model–View–ViewModel) est un pattern d'architecture logicielle utilisé principalement pour les interfaces utilisateur (applications desktop, web ou mobile).

- Il a été popularisé notamment par les technologies de data binding comme Microsoft Windows Presentation Foundation (WPF) et Angular.

- Son objectif est de séparer clairement la logique métier de l'interface graphique, tout en facilitant la synchronisation automatique entre les données et la vue.

- **Model**
    - représente les données et la logique métier
    - contient : les entités, les règles métier et l'accès aux données (DAO, BD, …)
- **View**
    - affiche les données
    - capture les actions de l'utilisateur (clic, saisie, gestes tactiles)
    - Exemples de vues: interface JavaFX, page HTML, interface Android
- **ViewModel**
    - l'élément central du MVVM
    - prépare les données pour la vue
    - contient la logique de présentation
    - expose les données sous forme observable pour la vue

- La View est liée automatiquement au ViewModel grâce au Data Binding
    - Data Binding = liaison des données
    - Permet de lier automatiquement les données du ViewModel avec la View.
    - Permet de synchroniser automatiquement les données entre la View et le ViewModel.
    - La manière d'implémenter le Data Binding dépend de la technologie utilisée.
      Les cas les plus courants sont : JavaFX, Angular, WPF, Android, …

- Si l'on n'utilise pas un framework comme JavaFX ou Angular, qui fournissent un Data Binding natif, il est toujours possible d'implémenter un Data Binding en Java pur en utilisant le pattern *Observer*.
    - le ViewModel possède des propriétés observables (l'objet observé est la propriété dans le ViewModel)
    - la View s'abonne aux changements (L'observer est la View)
    - lorsque la valeur change, la View est automatiquement notifiée.

Implémentation du pattern MVVM en Java : Projet Gestion de tâches en MVVM

- Vous devez développer une application simple de gestion de tâches (To-Do List) en respectant strictement le pattern MVVM.
- L'application fonctionnera en mode console.
- Simulation du Data binding: pattern de conception Observer

☞ **Travail noté**

- Utilisez JavaFX ou Angular + MVVM pour développer l'application To-Do List.