Interfaces Hommes/Systèmes - TD 2

‡ ‡ ‡

Architectures IHS & Logicielles

I - Un pas vers les architectures

Notre interface maintenant construite, nous allons nous intéresser à la liaison avec l'application...

Exercice I.1

Liaison avec l'application

Dans un premier temps, nous allons créer la structure de note projet à l'aide de l'outils *maven*.

1. Suivre les directives de création de votre gentil animateur de TD ...

Ceci étant fait, réalisons l'application assurant la conversion de devises :

2. Dans le projet *convertisseur-devises-business*, implémenter l'application
3. Dans le projet *convertisseur-devises-business-api*, créer l'API de votre application.

Il ne reste maintenant qu'à "brancher" notre interface utilisateur :

4. Dans le projet *convertisseur-devises-ui*, ajouter l'implémentation de l'interface graphique créée dans l'exercice précédent.
5. Utiliser l'interface de l'application pour implémenter le calcul de conversion en fonction des actions de l'utilisateur.
6. Tester.

II - Modification du cahier des charges

Nous reprenons ici la réalisation d'une application permettant de convertir des Euros en Dollars et inversement. Nous avons vu les différents défauts de la dernière version réalisée. Voyons comment améliorer les choses, en mettant en place les patterns d'architecture vus en cours.

Définissons tout d'abord de manière plus précise le *dialogue* proposé à l'utilisateur :

- si le champ d'entrée est vide, le choix du sens de conversion n'est pas disponible ; il le devient dès qu'un chiffre est saisi dans ce premier ;
- lorsque le clic sur le bouton a été effectué, champ texte d'entrée, choix du sens de conversion et bouton de conversion ne sont plus accessibles ;
- l'enfoncement de la touche `A` remet l'interface dans son état initial.

III - Architecture MVC multi-composants

Nous choisissons ici de mettre en place une architecture MVC multi-composants qui sera composée :

→ d'un composant champ texte (instancié 2 fois il nous servira pour les champs d'entrée et de sortie)
  - nous pourrions, avec un peu de courage, créer un champ composant champ texte n'acceptant que les numériques !

→ d'un composant action contenant le choix de conversion et le bouton de conversion

→ d'un composant global composé des précédents et en lien avec l'application.

Exercice III.1

UI du convertisseur de devises en MVC

1. Implémenter le composant action et le tester
2. Implémenter le composant champ texte (numérique) et le tester.
3. Implémenter le composant global.
4. Tester. Votre application maintenant terminée !

IV - Architecture MVP multi-composants

Pour nous permettre de toucher du doigt les différences notables entre l'architecture MVC utilisée précédemment et l'architecture MVP multi-composants, nous allons maintenant ré-implémenter l'interface graphique en utilisant ce pattern d'architecture.

Exercice IV.1

UI du convertisseur de devises en MVP

En vous basant sur le même *dialogue* que dans l'exercice précédent, proposer une version de l'interface graphique du convertisseur en respectant l'architecture MVP multi-composants :

1. Implémenter le composant action et le tester
2. Implémenter le composant champ texte (numérique) et le tester.
3. Implémenter le composant global.
4. Tester. Votre application maintenant terminée !
5. Conclure sur les différences majeures des architectures MVC VS MVP.

V - Mise en place de la couche données (DAL)

Fort de la dernière version très aboutie de notre application de conversion de devises, nous souhaitons que les taux de change des devises soient enregistrés dans une base de données afin d'en faciliter la mise à jour.
Ceci permettra également à l'interface graphique de proposer toutes les possiblités enregistrées dans cette base.

Exercice V.1

Réalisation de la couche DAL

Au travail...

1. Ajouter le projet qui permettra l'implémentation de cette couche comme module du projet principal.
2. Ajouter à ce nouveau projet la dépendance `sqlite-jdbc`. Ceci permettra de rapidement mettre en place une base de données fonctionnant de mabnière très similaire à un SGBD classique.

Prenons quelques instants pour parler du *Pattern DAO* dont un diagramme de classes UML est présentée figure 1 :

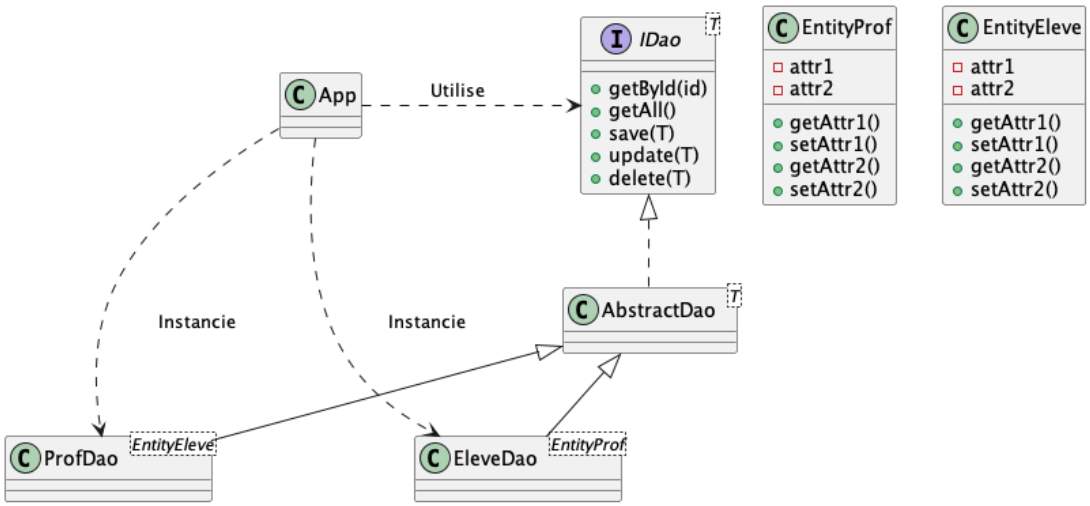

Figure 1 – Pattern DAO

3. Ajouter l'implémentation de cette nouvelle couche.
4. Tester depuis la classe `App` de votre projet.

Exercice V.2

Le grand final ! ! !

1. Utiliser les fonctionnalités de cette nouvelle couche pour proposer, au sein de la partie « métier » un nouveau convertisseur « générique ».
2. Modifier votre interface de telle sorte à tirer tous les bénéfices de cette dernière évolution.

‡ ‡ ‡