TP2 : Les capteurs
Programmation mobile

Mohamad Satea Almallouhi - Tony Nguyen
M1 Génie Logiciel
Faculté des Sciences
Université de Montpellier.

5 mars 2024

Résumé

Nous avons réalisé une application Android en Java afin de démontrer l'utilisation des capteurs intégré.

Table des matières

To Do
— diagram use case
— diagram sequence for synchronisation
— diagram state machine w/ tikz library to describe save function
— add code picture
— add smartphone picture
— ask Satea for part 2 & 3 explanation
— !!! make an .apk file for easy install, no compilation !!!
— diagram class observer DONE
— diagram class observer specific (Fragment Manager)

Introduction

Dans le cadre de l'Unité d'Enseignement Programmation Mobile, nous allons voir comment utiliser des fragments et la persistance des données.

Les sections du rapport suit les exercices.

Démonstration vidéo

1 Fragment

Un fragment est une portion réutilisable de l'interface utilisateur de notre application. Ils sont utiles car ils sont modulaires et réutilisables.

Dans cette section, nous allons voir comment les créer, les afficher et comment les faire communiquer entre eux.

1.1 Création

Afin de créer un nouveau fragment, il nous suffit d'hériter de la classe de base Android : "*Fragment*".

Pour spécifier le comportement du nouveau fragment, il nous suffira de masquer la méthode **onCreateView()**.

Dans ce TP, nous avons créer 2 fragments, **UserInputFragment** et **DisplayFragment**.

(UserInputFragment, DisplayFragment extends Fragment)

1.2 Ajout

Pour pouvoir les afficher, nous avons créer une activity et son layout vide.

Nous avons ensuite, ajouté une balise **fragment** pour chaque fragment crée. Il faut remarquer 2 attributs important.

L'attribut **android :name** associe cette balise .xml à aux classe fragment crée dans la sous-section précédente.

L'attribut **tools :layout** associe cette balise au layout indiqué.

Nous savons à présent comment créer un fragment, et l'afficher.

tag fragment - attribut name pour la classe et attribut layout pour la vue
Start_toStartOf Top_toTopOf ...

1.3 Couper l'écran en 2

C'était trop dur.

1.4 Communication entre fragment

Avant la communication, il est nécessaire de transmettre notre *ModelView* contenant les différentes MutableLiveData.

Une fois le model placé dans le *Bundle*, nous pouvons envoyer le message. Pour faire cela, nous utilisons la méthode **setFragmentResult(String requestKey, Bundle result)** pour envoyer les données (contenus dans le bundle avec requestKey une chaine de charactères fixés à l'avance par le programmeur, le fragment receveur devra utiliser la même requestKey).

Le message est maintenant envoyé, nous allons voir comment le recevoir.

Il sera nécessaire d'utiliser la méthode **setFragmentResultListener()** de la classe *FragmentManager*.

L'argument listener sera une classe anonyme qui hérite de *FragmentResultListener* dans laquelle on masque la méthode **onFragmentResult()**. Si nous avons bien utilisé les même requestKey, alors l'argument bundle contiendra les bonnes données. À partir de là, nos fragments possède la même instance du model.

1.5 La synchronisation et mise à jour

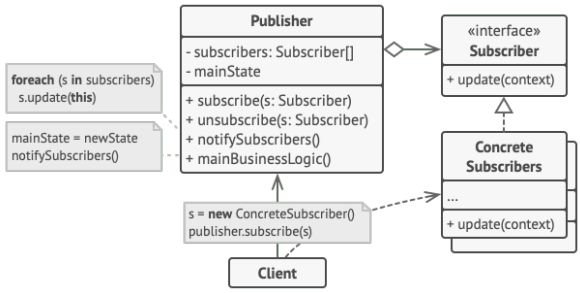
Lors de la détection d'un changements, nous souhaitons publié le changement et mettre à jour la vue. Pour cela, nous allons accéder aux différents *MutableLiveData* dans notre *ModelView* : *UserInputViewModel*. Une fois fait, nous publions simplement la nouvelle valeur avec la méthode **postValue()**. Nous envoyons les valeurs lorsque l'utilisateur appuie sur "Submit" ou après qu'il ait selectioné l'option "Synchronise" puis modifie le texte.

Afin d'être notifié des plublications, il faut s'abboner aux MutableLiveData et les **observer**.

Receiving
get the MutableLiveData from ViewModel and observe
Quand l'observateur est notifié, la méthode **onChanged()** est appelé et la nouvelle valeur est placé dans une $<TextView>$.

1.6 Observer

Voici un rappel du diagramme de classe du patron de conception *Observateur*



Il faut remarquer que ce design pattern est essentiel dans ce tp pour implémenter la communication entre les fragments. On y retrouve les méthodes de souscriptions, notifications et mises à jour.

On identifie les méthodes de notifications comme **setFragmentResult()** et **postValue()**.

Les méthodes de mises à jour correspondendt à **setFragmentResultListener()** et **observe()**.

2 Persistance

Nous allons maintenant nous intéressé à la sauvegarde sur le long terme.

2.1 Sauvegarde

Lors d'un clique sur le boutton *"Save"*, une écriture sur le stockage local du smartphone est enclenché.

La méthode **saveJsonToFile()** se chargera d'écrire dans le fichier *test.json*.

2.2 Saisie automatique

3 Réseau

Cet exerice n'a pas été réalisé.

4 Service

Après qu'une écriture sur le stockage du téléphone a été faite, il nous est possible de récupérer le contenu de ce fichier et de l'afficher à l'utilisateur.

À l'appuie du boutton "load", le service est démaré. *FileDownloadService* va simplement lire le fichier de sauvegarde prédéfini *test.json*.

Il va ensuite, diffuser les valeurs à travers un le *LocalBroadcastManager*.

Dans le DisplayFragment, Nous nous mettons à l'écoute du broadcast.

À la reception du broadcast, le text est écrasé par les différentes valeurs contenus dans le fichier.