UE TAC
M1 Informatique

Cours : Jean-Claude TARBY

TDs : Cédric DUMOULIN, Guillaume FORTIN, Jean-Claude TARBY

Université de Lille
FACULTÉ DES SCIENCES ET TECHNOLOGIES

Cours 7
RxJava, RxAndroid

ReactiveX

- http://reactivex.io/
- ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming
- Utilisé dans le TD 5 et 6
  - API Google Books
- Mais utilisable pour bcp d'autres choses (gestion de clics…)

- Programmation **réactive**

Base classes

- RxJava 3 features several base classes you can discover operators on:
  - io.reactivex.rxjava3.core.**Flowable**: 0..N flows, supporting backpressure
  - io.reactivex.rxjava3.core.**Observable**: 0..N flows, no backpressure,
  - io.reactivex.rxjava3.core.**Single**: a flow of exactly 1 item or an error,
  - io.reactivex.rxjava3.core.**Completable**: a flow without items but only a completion or error signal,
  - io.reactivex.rxjava3.core.**Maybe**: a flow with no items, exactly one item or an error.
- Doc exemple :
  - http://reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html
  - http://reactivex.io/documentation/operators/just.html
- Arbre de décision pour les opérateurs :
  - http://reactivex.io/documentation/operators.html

RxAndroid

- https://github.com/ReactiveX/RxAndroid

```
dependencies {
    implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
    implementation 'io.reactivex.rxjava3:rxjava:3.0.0'
}
```

Exemples pour commencer

- https://gitlab.com/m1-ue-tac/rxjava-rxandroid/just-map-et-flatmap
  - Création d'un flux avec écouteur + transformations
    - Création d'un flux Observable et d'un écouter
    - + opérateur 'map'
    - + opérateur 'flatmap'

```
this.disposable = this.getObservable()  // renvoie un Observable de type String qui contient juste la chaine 'Cool !'
        .map(getFunctionUpperCase())    // transforme l'observable précédent en majuscules
        .flatMap(getSecondObservable()) // enchaine sur le second Observable en passant le premier en paramètre
automatiquement
        .subscribeWith(getSubscriber());// déclenche la connexion sur le flux et donc le onNext...
```

Exemples pour commencer

- https://gitlab.com/m1-ue-tac/rxjava-rxandroid/exemples-de-observables
  - Les différents flux ‘observables’ : Observable, Flowable, Single, Maybe, Completable, CompletableAndThenObservable

```
/**
 * Petits exemples simple de RxJava pour montrer les différents observables.
 * ATTENTION ! Seules les méthodes testFlowable... et testObservable... sont correctement
 * écrites pour Android. Les autres méthodes sont écrites 'à la Java' et pas pour Android ! Elles
 * fonctionnent ici, mais dans le cadre d'une 'vraie' application Android, elles pourraient
 * faire crasher l'appli car mauvaise gestion des threads (dont celui de l'UI) !
 */
```

```
// pour RxJava3
implementation "io.reactivex.rxjava3:rxjava:3.0.0"
// pour RxAndroid
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
```

Un site parmi d'autres…

- https://www.androidhive.info/RxJava/tutorials/ (bon, mais date de 2018… passage à RxJava3 depuis)
  - https://www.androidhive.info/RxJava/rxjava-understanding-observables/
    - Explications très claires sur les observables, les schedulers, les CompositeDisposable, etc.
  - https://www.androidhive.info/RxJava/android-getting-started-with-reactive-programming/
    - Reactive programming = ?
    - ReactiveX = ?
    - RxJava
    - RxAndroid

LiveData ou Rx ?

- Comme vous voulez…
  - Rx
    - peut gérer les MAJ de data de bout en bout (API → IHM)
    - Offre + de possibilités que LiveData (pour filtrage, transformation…)
    - MAIS …. Ne gère pas le cycle de vie !
  - LiveData
    - Gère le cycle de vie
    - Offre quelques facilités pour MAJ des data (transformation, switch…)
    - Mise à jour automatique
    - MAIS… ne gère pas les appels asynchrones (sur API par exemple)
  - Rx + LiveData
    - Rx pour gestion data/API et remontée jusqu'à « data repository » par exemple
    - LiveData : flux Rx → « data repository » → LiveData (pour ViewModel et IHM)

    ➔ système + « souple », + « simple » dans les mises à jour,…

    ➔ + de code à comprendre/écrire

Cf. https://github.com/alinhayati/android-architecture qui montre différentes solutions (+ liens sur articles explicatifs) à partir d'un même exemple, mais écrit en **Kotlin**.

MVVM, MVVM avec LiveData, MVVM avec RxJava, MVVM avec LiveData et RxJava, …

Pour l’évaluation…

- TD 5 et 6 originels (Github de Maxime Savary-Leblanc)
  - « complexes » : (fausse) injection de dépendance
  - Vous pouvez faire + simple (pas d’injection de dépendance par exemple)
    - Cf. les TDs
    - Faire tout en RxJava, ou en mélangeant avec LiveData par exemple
- Pour évaluation
  - Pas besoin de faire de l’injection de dépendance (sauf si vous voulez)
    - Possible de faire un pattern Singleton ou utiliser la classe Application
    - Si vous voulez, vous pouvez utiliser Dagger ou Hilt
  - Pour le reste, cf. cahier des charges donné au début de l’année