ESIEE PARIS

Année 2019-2020
Semestre 2

Rapport Intermédiaire d'avancement de Projet Androïd

Avancement du 05/04/2020

Projet réalisé par :

MONTANI Maÿlis

Lien du Repos Github :

https://github.com/mayisu7798/ChuckNorrisJokesApp

Projet encadré par :

DUPONCHEL Nicolas

Partie III - 1 : Display a single Joke

Afin d'avoir une liste que je puisse modifier dans la classe `JokeAdapter`, il a fallu que je change le type de la liste passée en paramètre de la classe. Je l'ai modifiée en `MutableList<Joke>`.

```
class JokeAdapter (private var listJokes : MutableList<Joke>) :
```

Dans cette classe, j'avais dans un premier temps écrit la fonction "`setJokes`" en passant en paramètre une `List<Joke>` qui viendrait en remplacement de l'attribut `listJokes`. J'ai donc modifié cette fonction pour qu'elle ne fasse qu'ajouter une nouvelle `Joke` à la `MutableList<Joke>`.

```
fun setJokes(Joke : Joke)   {
    this.listJokes.add(Joke)
    this.notifyDataSetChanged()
}
```

Dans le fichier `MainActivity.kt`, j'ai ensuite modifié la liste que je passais en paramètre de mon `JokeAdapter` pour lui passer une `MutableList<Joke>`.

```
val listeOfJokes : MutableList<Joke> = mutableListOf()
//val adapteur = JokeAdapter(StaticList.list.toJokeList())
val adapteur = JokeAdapter(listeOfJokes)
```

Dans mon `subscribeBy(...)` j'ai ensuite ajouté la `Joke` obtenue à ma liste de `Joke` de `JokeAdapter`. En lançant mon code, cela fonctionnait bien que je n'ai pas ajouté le `observeOn(...)` mais dans un soucis de respect des consignes, je l'ai quand même ajouté.

```
val resultSubscribe = joke.subscribeOn(Schedulers.io()).observeOn(AndroidSchedulers.mainThread())
    .subscribeBy(
        onError =   {   println("Le Single<Joke> renvoie une erreur")   },
        onSuccess = {   joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
                                adapteur.setJokes(joke)
                    }
    )
```

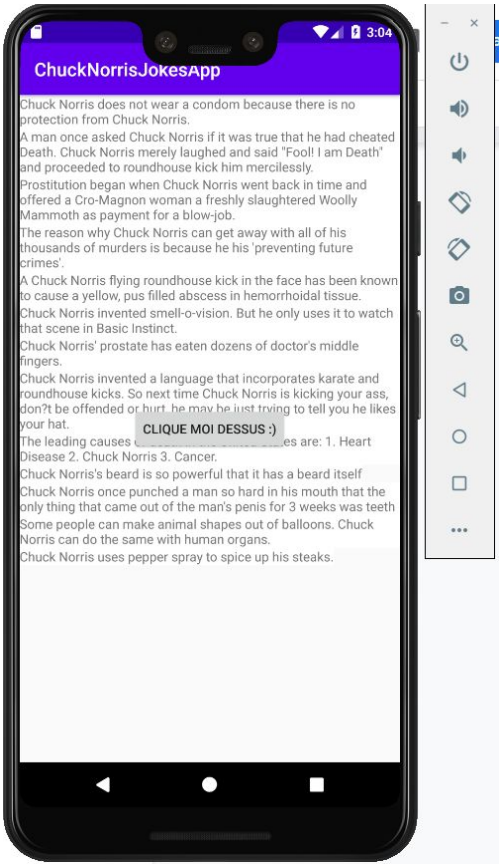
Dans le fichier `activity_main.xml`, j'ai ajouté un tag `<Button/>` permettant d'ajouter un bouton sur mon application.

```
<Button
    android:id="@+id/Button"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/CliqueDessus"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Dans le fichier `MainActivity.kt`, j'ai ensuite ajouté ces quelques lignes après avoir modifié les "`val`" en "`var`" pour mes variables "`joke`" et "`resultSubscribe`" :

```
val button : Button = findViewById(R.id.Button)
button.setOnClickListener { it: View!
    joke = jokeService.giveMeAJoke()
    resultSubscribe = joke.subscribeOn(Schedulers.io()).observeOn(AndroidSchedulers.mainThread())
        .subscribeBy(
            onError =   {   println("Le Single<Joke> renvoie une erreur")   },
            onSuccess = {   joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
                adapteur.setJokes(joke)
            }
        )
}
```

Finalement, mon application fonctionne correctement, en dehors du bouton qui est au dessus du layout d'affichage des Jokes, ce qui donne un rendu assez moyen, mais ça reste fonctionnel :



Temps : ~ 1 heure

Partie III - 2 : Add a loader

Dans un premier temps, j'ai créé ma `ProgressBar` dans mon `activity_main.xml` pour l'ajouter dans mon application. Le paramètre "`gone`" permet de positionner la visibilité de la `ProgressBar` sur "inexistante" comme si elle n'avait même pas été insérée dans le layout.

```
<ProgressBar
    android:id="@+id/ProgressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="50dp"
    android:minWidth="50dp"
    android:visibility="gone"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="@id/Button" />
```

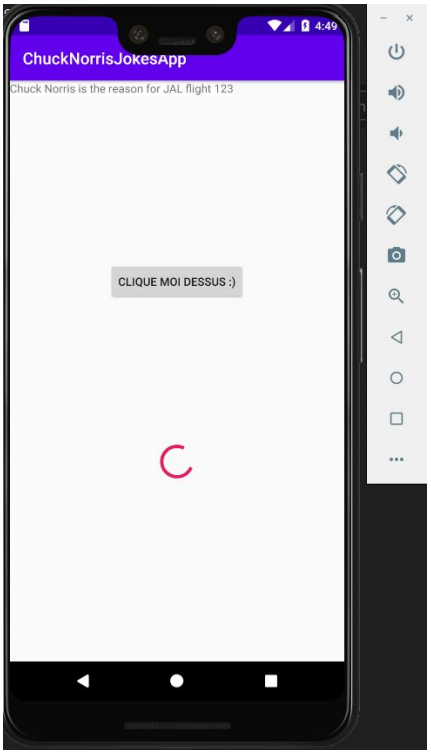
J'ai d'abord essayé de mettre les contraintes sur la visibilité dans le code du `MainActivity.kt` en encadrant la ligne du `resultSubscribe` sur laquelle j'avais ajouté un délais. Cependant, cette méthode ne fonctionne pas puisque l'exécution du `resultSubscribe` se fait sur un autre Thread, ce qui ne laisse pas le temps à la `ProgressBar` de s'afficher comme je le souhaite.

```
val progressBar : ProgressBar = findViewById(R.id.ProgressBar)
button.setOnClickListener { it: View!
    joke = jokeService.giveMeAJoke()
    progressBar.visibility = VISIBLE
    resultSubscribe = joke.delay( time: 2, TimeUnit.SECONDS)
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribeBy(
            onError =   {   println("Le Single<Joke> renvoie une erreur")   },
            onSuccess = {   joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
                adapteur.setJokes(joke)
            }
        )
    progressBar.visibility = INVISIBLE
```

Il a alors fallu que je change la façon dont je procédais, en utilisant une autre façon d'implémenter la subscription. Ici on utilise les `.doOnError`, `.doOnSuccess`, `.doOnSubscribe` et `.doAfterTerminate` pour indiquer les actions à réaliser. Ces deux dernières fonction permettent au Thread courant d'effectuer des actions au lancement et à la terminaison du Thread dédié à la subscription. On termine par un `.subscribe()` pour retourner un type `Disposable` (que nous avions à l'aide de la fonction `subscribeBy()` avant le changement).

```
val progressBar: ProgressBar = findViewById(R.id.ProgressBar)
button.setOnClickListener { it: View!
    joke = jokeService.giveMeAJoke()
    resultSubscribe = joke.subscribeOn(Schedulers.io())
        .delay( time: 1, TimeUnit.SECONDS, AndroidSchedulers.mainThread())
        .doOnError { println("Le Single<Joke> renvoie une erreur") }
        .doOnSubscribe { progressBar.visibility = VISIBLE }
        .doOnSuccess { joke ->
            println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
            adapteur.setJokes(joke)
        }
        .doAfterTerminate { progressBar.visibility = INVISIBLE }
        .subscribe()
}
```

Au final, on obtient bien ce qu'on souhaite, c'est à dire une ProgressBar inactive tant qu'on appuie pas sur le bouton et que le traitement demandé n'est pas effectué (capture réalisée à l'appui sur le bouton) :



Note : J'ai eu beaucoup de mal à comprendre ce qui n'allait pas avec ma première approche mais dès que j'ai compris le problème des Threads, tout allait mieux.
Temps : ~ 2 heures
Commit : Affichage d'une Joke seule et Ajout d'une ProgressBar

Partie III - 3 : Make the call for multiple jokes with Observable

Pour cette partie, il fallait simplement ajouter `.repeat(10)` dans les conditions du `resultSubscribe`. Ajouter cette condition rend le `.doOnSuccess{...}` faux, il faut alors le remplacer par `.doOnNext{...}`. Le reste du code reste strictement identique.

```
resultSubscribe = joke.subscribeOn(Schedulers.io())
    .delay( time: 1, TimeUnit.SECONDS, AndroidSchedulers.mainThread())
    .repeat( times: 10)
    .doOnError { println("Le Single<Joke> renvoie une erreur") }
    .doOnSubscribe { progressBar.visibility = VISIBLE }
    .doOnNext { joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
                adapteur.setJokes(joke) }
    .doAfterTerminate { progressBar.visibility = INVISIBLE }
    .subscribe()
```

Temps : ~ 20 minutes
Commit : Répétition 10 fois des appels de Joke par l'appui sur le bouton

Partie III - 4 : Reload new jokes

Pour retirer le bouton, il suffisait de retirer le tag `<Button/>` du fichier `activity_main.xls` et de retirer sa création et son appel de la fonction à `setOnClickListener()` dans le `MainActivity.kt`. En l'état, l'application fonctionne et affiche 10 Joke avec un délais d'une seconde entre chaque.

On ajoute ensuite un attribut fonction à notre classe `JokeAdapter` :

```
class JokeAdapter(private var listJokes: MutableList<Joke>, val onBottomReached: (JokeAdapter) -> Unit)
    : RecyclerView.Adapter<JokeAdapter.JokeViewHolder>()
```

Dans un premier temps, j'avais essayé d'écrire la fonction lambda comme une variable dans le fichier `MainActivity`. Cette méthode ne fonctionnait pas parce que je ne pouvais pas créer un `adapter` sans ma fonction `onBottomReached` et la fonction `onBottomReached` avait besoin de l'`adapter` pour s'exécuter. (à ce moment là je n'avais pas encore mis de paramètre dans ma fonction `onBottomReached`).

Il a alors fallu que je change la façon dont je donnais la fonction à mon `adapter`. J'ai utilisé cette syntaxe :

```
val joke: Single<Joke> = jokeService.giveMeAJoke()
val adapteur = JokeAdapter(listeOfJokes){ jokeAdapter : JokeAdapter ->
    val resultSubscribe = joke.subscribeOn(Schedulers.io())
        .delay( time: 100, TimeUnit.MILLISECONDS, AndroidSchedulers.mainThread())
        .repeat( times: 10)
        .doOnError { println("Le Single<Joke> renvoie une erreur") }
        .doOnSubscribe  {   progressBar.visibility = VISIBLE    }
        .doOnNext { joke ->
            println("Yaaay on peut utiliser la Jooooke : ${joke.value}")
            jokeAdapter.setJokes(joke)

        }
        .doAfterTerminate   {   progressBar.visibility = INVISIBLE      }
        .subscribe()
    disposable.add(resultSubscribe)
}
recycler.adapter = adapteur
```

Il fallait ensuite qu'on indique à l'`adapter` quand nous devions appeler cette fonction. J'ai beaucoup (beaucoup beaucoup) cherché une solution pour ce problème. Ma première idée était de trouver quelque chose qui nous indique quand nous arrivons à la fin du scroll possible, mais je ne l'ai pas trouvé à ce moment car je cherchais dans `RecyclerView.Adapter` et pas dans `RecyclerView`. J'ai essayé beaucoup de choses que j'ai trouvé, j'ai fais beaucoup de tests. Sans succès. J'ai finis par comprendre que je cherchais au mauvais endroit puis j'ai trouvé cette solution, qui fonctionne :

```
adapteur.onBottomReached(adapteur)
recycler.viewTreeObserver.addOnScrollChangedListener ( ViewTreeObserver.OnScrollChangedListener {
        if (!recycler.canScrollVertically( direction: 1)) {
            adapteur.onBottomReached(adapteur)
        }
    }
)
```

Note : J'ai vraiment perdu énormément de temps sur cette question. Je ne comprenais pas ce que je devais faire, j'ai cherché des solutions avec des CustomListener etc mais je n'arrivais pas à les faire coller à notre consigne avec la fonction lambda en paramètre.
Temps : ~ 4 heures
Commit : Charger de nouvelles Jokes en scrollant vers le bas

Partie III - 5 : Conclusion

Note : Je m'arrête ici pour aujourd'hui.
Total temps : ~ 7 heures 20 minutes