RxJava: Multi-Threading in Android

What Is RxJava ?

RxJava is a Java VM implementation of Reactive Extensions: a library for composing asynchronous and event-based programs by using observable sequences.

we can define it as an API for asynchronous programming with observable streams. It is a combination of the best ideas from the observer pattern, the iterator pattern, and functional programming.

Easy multi-threading

For asynchronous work, thread management is crucial. In many situations, while a task is being performed, we encounter a situation of communicating between a background thread and the main thread.

Simple error handling

Errors are the most frustrating thing for developers. While performing a lot of complex, asynchronous operations, we encounter errors in many places.

Avoid the callback hell:

At any point, if you have made nested network calls, you may know what the pain will be of handling them. But Rx has many operators to resolve these kinds of issues very easily.

Formula

`Rx= SCHEDULERS + OBSERVABLE + OBSERVER`

Observable

In RxJava, Observables are the source that emits data to the Observers. For Observers to listen to the Observables, they need to subscribe first.https://miro.medium.com/max/1400/1*dk5LNuA3lmBuU7SwQfjITQ.png

`Observable.OnSubscribe` is an interface that defines the action to be taken when a subscriber subscribes to the Observable. The subscribe method would only run when an Observer is subscribed to the Observable.

Let’s create a simple Observable.

```
val observableObject =
Observable.create(ObservableOnSubscribe<Int>{

    if(!it.isDisposed){
        it.onNext(10)
    }
    if(!it.isDisposed){
        it.onNext(20)
    }
    if(!it.isDisposed){
        it.onNext(30)

        it.onComplete()
    }
} )
```

Observer

Observers are the components that consume the data that is emitted by an observable. In Rx, Observers subscribe to the observable using the `subscribe` method to receive the data emitted by the observable.

There are three basic methods that we should know to understand the Observer.

- `onNext()`: Used to receive data

when the observable emits the next item.

- `onError()`: Triggered when an error occurs.
- `onComplete()`: Called after the last item is emitted. Basically when the work is completed.
- Lets create observer instance

```
val observerInstance = object :
io.reactivex.Observer<Int> {

    override fun onComplete() {
        println("onComplete")
    }

    override fun onSubscribe(d: Disposable) {
        println("onSubscribe")
    }

    override fun onNext(t: Int) {
        println("onNext " +t)
    }

    override fun onError(e: Throwable) {
        println("onError "+e.localizedMessage)
    }

}
```

Add dependencies to app-level `build.gradle`.

```
//RxJava
implementation
'io.reactivex.rxjava2:rxjava:2.2.8'
//RxAndroid
implementation
'io.reactivex.rxjava2:rxandroid:2.1.1'
//ViewModel
implementation
'androidx.lifecycle:lifecycle-
viewmodel-ktx:2.2.0'
```

    1. `subscribe()`: This method has two parts, success and failure blocks, where we can handle each case as per the requirement.
    2. `subscribeOn()` and `observeOn()`: `subscribeOn()` is used to specify on which thread the Observable should perform its work and `observeOn()` is used to specify on which thread the Observable should emit the items.
3. `onCleared()`: This is the overridden method of `ViewModel` where we can dispose of all the disposables. This method is called when `ViewModel` is getting destroyed.