Android Advance 2
Lesson 4
Reactive for Android

Outline

I. Introduction to ReactiveX

II. Implement ReactiveX for Android

III. Example

I. Introduction to ReactiveX

1. What is reactiveX

- ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences.
- It extends the observer pattern to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety, concurrent data structures, and non-blocking I/O.

2. What is RxJava?

- RxJava is a reactive extension for the Java virtual machine
- Reactive programming is programming with asynchronous data streams.
- Functional Reactive Programming combines reactive programming and functional programming. We will using functional reactive programming when apply RxJava in Android

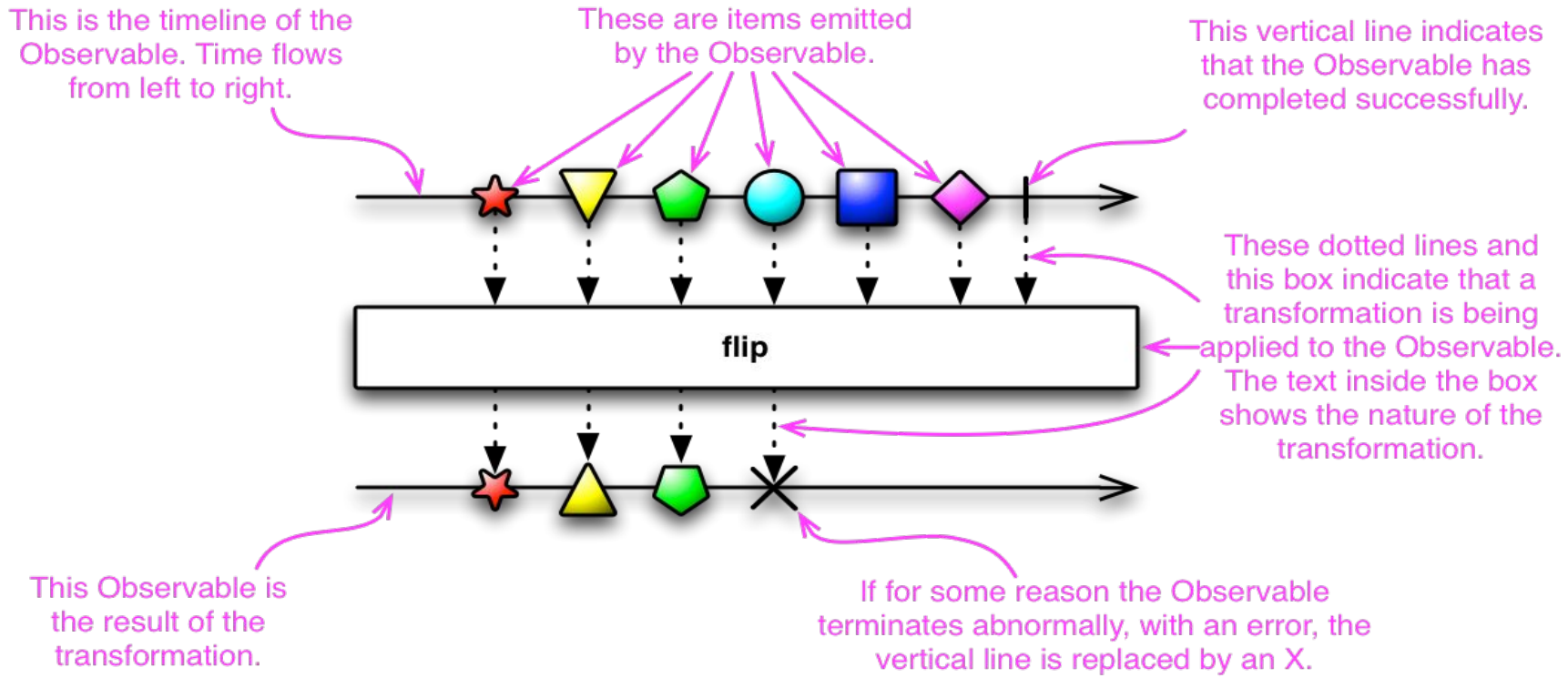
3. Why consider RxJava ?

- Make dealing with concurrency easy
- Exposes a more explicit way for declaring how concurrent operations should operate that make code a lot more concise and readable
- Surfaces errors sooner
- Helps reduce the need for state variables that can introduce bugs

4. Core RxJava construts

- **Observable:**
  - Contains all the heavy processing
  - Emit items (Objects, String, Integers etc,…)
  - Does not start emitting till someone subscribes
- **Subscribers:**
  - Consumes data
  - Has #onNext(…), #onComplete(…) and #onError(…)
- **Operator:**
  - Most powerful part about Observables is that you can *transform* them and perform functional style programming – *map()*, *debounce()*, *filter()*, *etc*
  - Transform Observable instances through Operators

5. Operator



6. RxAndroid

- Is an extension for RxJava built just for Android
- Provides bindings for Android
- *AndroidSchedulers* provides ready for use with the Android multi-threading system
- *ViewObservable* & *WidgetObservable* provides bindings for views, clicks, scroll, input etc…
- *AndroidObservable* provides facilities for working within the Android lifecycle. Bind to activity, fragment or even broadcasts.

7. Schedulers

Specify a Scheduler where the Observable should operate using *#subscribeOn(...)*, specify a Scheduler where the Observable should notify its observers using *#observeOn(...)*

Scheduler	purpose
`Schedulers.computation( )`	meant for computational work such as event-loops and callback processing; do not use this scheduler for I/O (use `Schedulers.io( )` instead); the number of threads, by default, is equal to the number of processors
`Schedulers.from(executor)`	uses the specified `Executor` as a Scheduler
`Schedulers.immediate( )`	schedules work to begin immediately in the current thread
`Schedulers.io( )`	meant for I/O-bound work such as asynchronous performance of blocking I/O, this scheduler is backed by a thread-pool that will grow as needed; for ordinary computational work, switch to `Schedulers.computation( )`; `Schedulers.io( )` by default is a `CachedThreadScheduler`, which is something like a new thread scheduler with thread caching
`Schedulers.newThread( )`	creates a new thread for each unit of work
`Schedulers.trampoline( )`	queues work to begin on the current thread after any already-queued work

II. Implement ReactiveX for Android

1. Config

- Add these line into file **build.gradle**

```
compile 'io.reactivex:rxandroid:1.2.1'
```

2. Create an observable

- Create an Observable using *#create(...)*
- Create an Observable from item/s using *#just(...)*
- Create an Observable from an *Iterable* using *#from(...)*
- Create an Observable that emits items given an interval using *#interval(...)*
- Create an Observable until subscription using *#defer(...)*

3. Subscribing to an Observable

```
Observable<Integer> o = Observable.just(1,2,3,4);
o.subscribe(new Observer<Integer>() {
    @Override public void onCompleted() {
        Log.d("Test", "In onCompleted()");
    }

    @Override public void onError(Throwable e) {
        Log.d("Test", "In onError()");
    }

    @Override public void onNext(Integer integer) {
        Log.d("Test", "In onNext(): " + String.valueOf(integer));
    }
});
```

```
Test  D  In onNext(): 1
      D  In onNext(): 2
      D  In onNext(): 3
      D  In onNext(): 4
      D  In onCompleted()
```

4. #*map(...)* operator

```
Observable.just(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).map(new Func1<Integer, Integer>() {
    @Override public Integer call(Integer integer) {
        return 2 * integer;
    }
}).subscribe(new Observer<Integer>() {
    @Override public void onCompleted() {
    }

    @Override public void onError(Throwable e) {
    }

    @Override public void onNext(Integer integer) {
        // Will receive 2,4,6,8,10...
    }
});
```

5. #filter(...) operator

```
Observable.just(1, 2, 3, 4, 5, 6, 7, 8, 9, 10).map(new Func1<Integer, Integer>() {
    @Override public Integer call(Integer integer) {
        return 2 * integer;
    }
}).subscribe(new Observer<Integer>() {
    @Override public void onCompleted() {
    }

    @Override public void onError(Throwable e) {
    }

    @Override public void onNext(Integer integer) {
        // Will receive 2,4,6,8,10...
    }
});
```

Exercise

- Continue implement project search users on github apply Retrofit using Observables of ReactiveX instead of “enqueue(Callback<T> callback)” of retrofit & Handler Error when no network connection.