**RXJAVA 2**

* **What is RXJAVA?**
* RxJava is used for *reactive* *programming*. In reactive programming, the consumer reacts to the data as it comes in. Reactive programming allows for event changes to propagate to registered observers.
* ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming.
* RxJava is a Java VM implementation of ReactiveX (Reactive Extensions): a library for composing asynchronous and event-based programs by using observable sequences.
* RxJava tries to be very lightweight. It is implemented as a single JAR that is focused on just the Observable abstraction and related higher-order functions.
* **Buildings block**

1. observables representing sources of data.
2. subscribers (or observers) listening to the observables.
3. A set of methods for modifying and composing the data.

* **Main components**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | observables | Observables are the sources for the data. Usually they start providing data once a subscriber starts listening. An observable may emit any number of items (including zero items). It can terminate either successfully or with an error. Sources may never terminate, for example, an observable for a button click can potentially produce an infinite stream of events |
| 2 | observers | It is also possible to convert the stream before it’s received by the observers.  Observer is the counter part of Observable which listens to the data emitted by Observable by subscribing. |
| 3 | subscriptions | A observable can have any number of subscribers. If a new item is emitted from the observable, the onNext() method is called on each subscriber. If the observable finishes its data flow successful, the onComplete() method is called on each subscriber. Similar, if the observable finishes its data flow with an error, the onError() method is called on each subscriber. |
| 4 | Observable | An Observable is the stream abstraction in RxJava. It is similar to an Iterator in that, given a sequence, it iterates through and produces those items in an orderly fashion. A consumer can then consume those items through the same interface, regardless of the underlying sequence.  Observable is an object which emits data. |

* **asynchronous programming**
* This allows simplifying the asynchronously processing of potential long running operations.
* It also provides a defined way of handling multiple events, errors and termination of the event stream. Reactive programming provides also a simplified way of running different tasks in different threads.
* It is also possible to convert the stream before it’s received by the observers. And you can chain operations,

**For example**

If an API call depends on the call of another API Last but not least; reactive programming reduces the need for state variables, which can be the source of errors.

* **RxJava 2 features several base classes you can discover operators on**

1. io.reactivex.Flowable : 0..N flows, supporting Reactive-Streams and backpressure
2. io.reactivex.Observable: 0..N flows, no backpressure
3. io.reactivex.Single: a flow of exactly 1 item or an error
4. io.reactivex.Completable: a flow without items but only a completion or error signal
5. io.reactivex.Maybe: a flow with no items, exactly one item or an error

***Working Generic RXJava (MVVM)***