**Research**

1. What is the difference in RxJava v1 and v2?

* RxJava v2 is under Maven coordinates (io.reactivex.rxjava2:rxjava:2.x.y) and classes are now accessible under io.reactivex.
* Users switching from 1.x to 2.x have to reorganize their imports carefully.
* Dependencies changed.
* onCompleted now became onComplete.
* compositeSubscription now became CompositeDisposable.
* Func1 is now Function and Fun2 is now BiFunction.
* Limit operator has been removed and in v2 now is called take operator.
* RxJava2 has better performance.
* RxJava2 has lower memory usage.
* Non-Backpressured Observable and the new Flowable (backpressured enabled) became available in RxJava2.
* onSuccess and onError have been redesigned from scratch and its architecture now follows the Reactive-Streams design.
* Maybe class is introduced as a new base reactive type.
* Its consumer type (rx.Single.SingleSubscriber<T>) has been changed from being a class that accepts rx.Subscription resources to be an interface io.reactivex.SingleObserver<T> that has only 3 methods:
* interface SingleObserver<T> {
* void onSubscribe(Disposable d);
* void onSuccess(T value);
* void onError(Throwable error);
* }

1. What is backpressure?

Backpressure in RxJava happens only when more results come in than it can be emitted. Backpressure comes into picture when you have an observable which emits items so fast that the consumer can’t keep up with the overflowing existence of emitted items. It requires system resources to handle backpressures. Control and management of the emitted and unconsumed items is what backpressure problems that needs to be dealt with. Some strategies in dealing with backpressure are:

* by reducing the number of items emitted by the observable using RxJava operators,
* by collecting and emitting items as a collection using RxJava operators, and
* by using the reactive pull strategy. In this strategy, subscriber requests require a number of items from observable by calling request().

1. Describe how the Observable subscriber pattern works?

The building blocks of the Observer-Subscriber pattern are:

* Observables represent sources of data. An observable emits items/data.
* Subscribers/observers listen to the observables. A subscriber consumes those items/data.
* A set of methods for modifying and composing the data.

Observables are the sources of data. They start providing/emitting data once a subscriber starts listening. An observable may emit 0 to any number of data and it can terminate successfully, with an error, or even may never terminate. An observable can have any number of subscribers. If a new item is emitted by the observable, the onNext() method is called on each subscriber. If a data flow is finished successfully by an observable, the onComplete() method is called on each subscriber. However, if the data flow is finished with an error produced by an observable, the error() method is then called on each subscriber.

1. What is the operators map and flatmap. How do they differ?

Map is an operator in RxJava whose goal is to tick the result of an observable and connects into a function. Flatmap is an RxJava operator that takes the results from one observable and connect it into another observable. The difference between the two is that Map connects the result of an observable into a function, whereas flatmap connects it to another observable. In other words, the items emitted by an observable are transformed by the Map by applying a function to each item, whereas flatmap transforms the items emitted by an observable into another observable. Another main difference is that the flatmap returns an observable itself, so it is used to map over asynchronous operations.