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

- RxJava v2 targets the Reactive-Streams SPI directly
- improved our memory consumption and performance considerably
- has better performance and low memory usage
- **Observable** doesn't care about backpressure, which greatly simplifies its design and implementation. It should be used to model streams that can't support backpressure by definition, e.g. user interface events.
- **Flowable** does support backpressure and has all the safety measures in place. In other words, all steps in the computation pipeline make sure you are not overflowing the consumer.

2. What is backpressure?

It is simply the process of handling a fast item producer. If an Observable produces 1.000.000 items per second how a subscriber which can handle only 100 items per second does process the items? The Observable class has an unbounded buffer size, it means it will buffer everything and pushes it to the subscriber, that's where you get the OutOfMemoryException. By applying Backpressure onto a stream, it'll be possible to handle items as needed, unnecessary items can be discarded or even let the producer know when to create and push the new items.

3. Describe how the Observer subscriber pattern works?

The Observer Pattern (also known as Publish-Subscribe Pattern) is a behavioral design pattern which

- defines a one-to-many relationship between objects such that, when one object changes its state, all dependent objects are notified and updated automatically.
- An object with a one-to-many relationship with other objects who are interested in its state is called the subject or publisher.
- Its dependent objects are called observers or subscribers.
- The observers are notified whenever the state of the subject changes and can act accordingly.
- The subject can have any number of dependent observers which it notifies, and any number of observers can subscribe to the subject to receive such notifications.

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

Map

transform the items emitted by an Observable by applying a function to each item

FlatMap

transform the items emitted by an Observable into Observables, then flatten the emissions from those into a single Observable

How do they differ?

In the Map operator applies a function of your choosing **to each item** emitted by the source Observable, and returns an Observable that emits the results of these function applications and the FlatMap operator transforms an Observable by applying a function that you specify to each item emitted by the source Observable, where that function returns an Observable that itself emits items. FlatMap then merges the emissions of these resulting Observables, emitting these merged results as its own sequence