FUNCTIONAL REACTIVE PROGRAMMING ON ANDROID

ZAC SIEGEL

LEAD ANDROID ENGINEER AT AKTA
@ZAC_SIEGEL

ÄKTA

DESIGN + INNOVATION

Reactive Systems... are repeatedly prompted by the outside world and their role is to continuously respond to external inputs

David Harel and Amir Pnueli

EVERYDAY ANDROID

» Network

» Database

» Filesystem

» Device State (device rotation, sensor updates)

» User Input (user input)

COMPLEXITY

- These operations are often dependent
  - Network -> DB -> UI
  - Error handling at each step
- Work needs to be done off the UI thread
  - Context and thread switching is hard

FRP allows us to combine, observe, and react to these operations

THINKING IN STREAMS

» Minimizes the use of state variables

» Reverses our thinking of traditional operations

ITERATIVE THINKING

» ITERATION - pull data from an array

```
for (Integer number : numbers) {
    // Do some work
}
```

REACTIVE THINKING

» OBSERVATION - the array pushes its data to us and we react

```
Observable.from(numbers)
          .subscribe({ num ->
             // Do some work
          });
```

ADVANTAGES

» Less to keep track of!

» Encapsulates code in a reusable manner

ADVANTAGES

» Observables are composable and have a fluent API

  » Similar to the Android animation API

```
view.animate()
    .x(10)
    .y(10)
    .duration(1000)
    .start();
```

ADVANTAGES

» Allows for combination of operations

CALLBACK SPAGHETTI HELL

```
apiService.login({
    apiService.getUserData({
        dbService.saveUserData({
            uiThread.updateUI({

            })
        })
    })
});
```

ADVANTAGES

» Allows for combination of operations

  » No callback spaghetti

```
apiService.login()
          .getUserData()
          .saveUserData()
          .subscribe(Subscriber {
             void onComplete(){}
             void onError(Exception e){}
             void onNext(Object o){}
           });
```

ADVANTAGES

» Explicit threading capability

  » Easily define where things get done

```
apiService.login() //Background thread
          .getUserData() //Background thread
          .saveUserData() //Background thread
          .subscribe(Subscriber { //UI thread
             void onComplete(){}
             void onError(Exception e){}
             void onNext(Object o){}
           });
```

EVERYDAY ANDROID

- A typical Android app
  - Network API Service
  - Database Service
  - Authentication Service

EVERYDAY ANDROID - PROBLEMS

» Rotation destroys everything

» All work must be done on different threads

» UI updates require us to be on the main thread

RXJAVA TO THE RESCUE

- Open source from Netflix
- Based on a spec from Microsoft
- JVM based implementation of FRP
  - language/platform specific contributions

BASIC BUILDING BLOCKS

- Observable
  - A single operation or function
  - Explicit threading (subscribing/observing)
  - If thread is undefined its synchronous

BASIC BUILDING BLOCKS

- Subscriber
  - An interface for updates
  - onNext(Object o) 1 to many times
  - onError(Exception e) 1 time
  - onComplete() 1 time

EXAMPLE

https://github.com/zsiegel/RX-GDG

COMMON OPERATORS

- `flatMap()`
- `merge()`
- `mergeDelayError()`
- `last()`

RTFM

The RxJava Github wiki is an incredible resource for understanding the power and functionality of Observables. READ IT!

To get the most out of RxJava you need to learn to combine and transform your Observables

QUESTIONS