Reactive Programming

RxAndroid in practice

Supercharge

Agenda

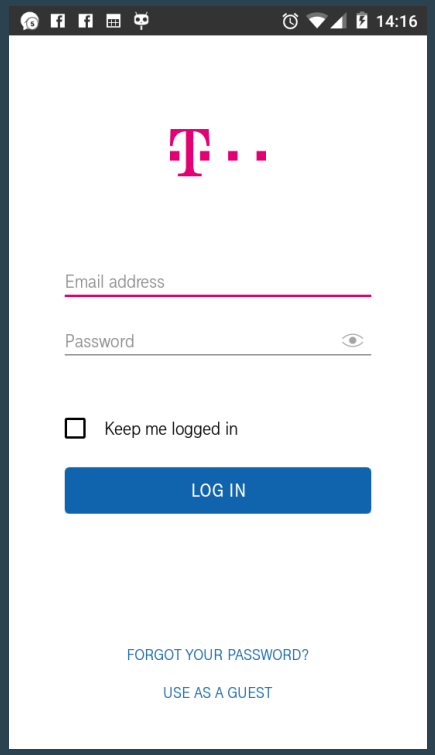
- Code
- Code
- Code
- Testing

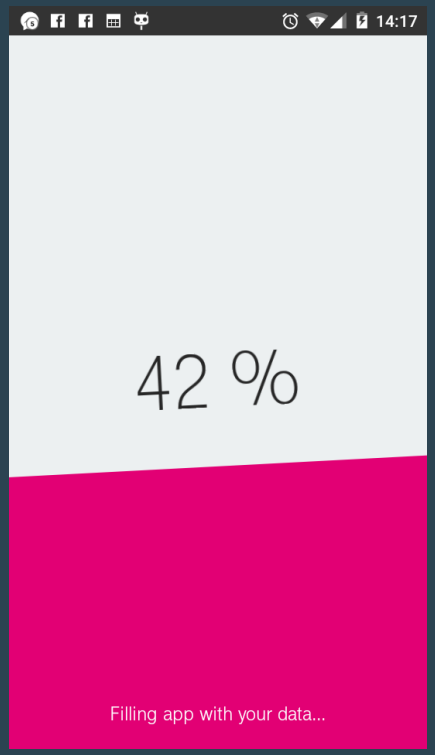
Introduction

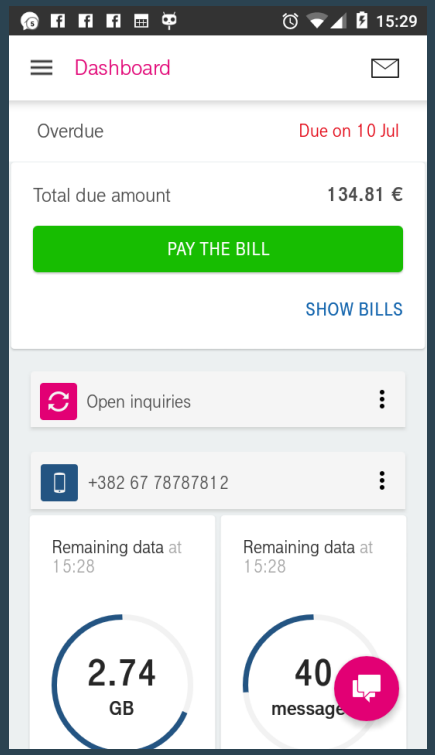
RxJava is a Java VM implementation of ReactiveX (Reactive Extensions): a library for composing asynchronous and event-based programs by using observable sequences.

The basic building blocks of reactive code are Observables and Subscribers. An Observable emits items; a Subscriber consumes those items.

Self-care app







Advanced Scenario

- Imagine that you have a login screen.

Advanced Scenario

- Imagine that you have a login screen.
- Your task is to validate the input fields and if everything is good, you enable the login button.

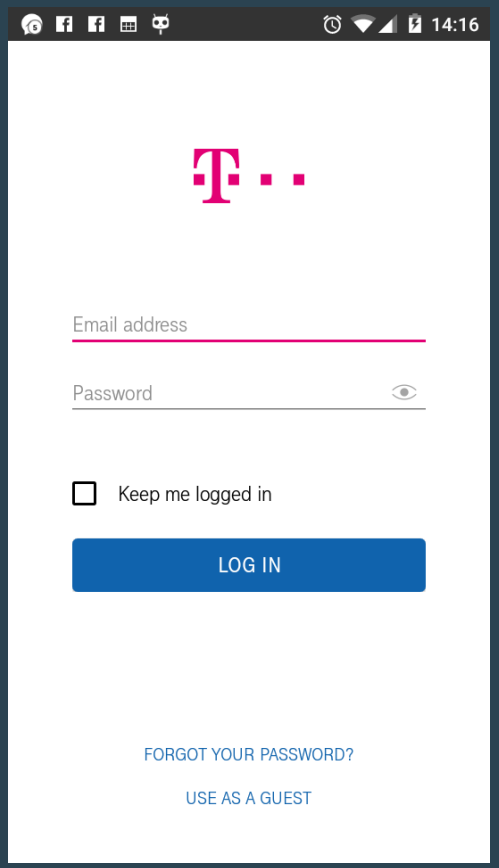
Advanced Scenario

- Imagine that you have a login screen.
- Your task is to validate the input fields and if everything is good, you enable the login button.
- After login, download all the data for the application.

Advanced Scenario

- Imagine that you have a login screen.
- Your task is to validate the input fields and if everything is good, you enable the login button.
- After login, download all the data for the application.
- For example: the user has services and bills. You can get it from different sources.

Login


14:16
Email address
Password
Keep me logged in
LOG IN
FORGOT YOUR PASSWORD?
USE AS A GUEST

Advanced Scenario

- Validate the input fields.

```
Observable.combineLatest(
    RxTextView.textChangeEvents(mEmailView),
    RxTextView.textChangeEvents(mPasswordView),
    (emailChange, passwordChange) -> {
        boolean emailOK = emailChange.text().length() >= 3;
        boolean passOK = passwordChange.text().length() >= 3;

        return emailOK && passOK;
    })
    .compose(bindToLifecycle())
    .subscribe(aBoolean -> mSignIn.setEnabled(aBoolean));
```

Advanced Scenario

- RxBinding provides binding for android's UI widgets.

```
Observable.combineLatest(
    RxTextView.textChangeEvents(mEmailView),
    RxTextView.textChangeEvents(mPasswordView),
    (emailChange, passwordChange) -> {
        boolean emailOK = emailChange.text().length() >= 3;
        boolean passOK = passwordChange.text().length() >= 3;

        return emailOK && passOK;
    })
    .compose(bindToLifecycle())
    .subscribe(aBoolean -> mSignIn.setEnabled(aBoolean));
```

Advanced Scenario

- Combine and evaluate the changes.

```
Observable.combineLatest(
    RxTextView.textChangeEvents(mEmailView),
    RxTextView.textChangeEvents(mPasswordView),
    (emailChange, passwordChange) -> {
        boolean emailOK = emailChange.text().length() >= 3;
        boolean passOK = passwordChange.text().length() >= 3;

        return emailOK && passOK;
    })
    .compose(bindToLifecycle())
    .subscribe(aBoolean -> mSignIn.setEnabled(aBoolean));
```

Advanced Scenario

- Use RxLifeCycle by Trello to properly handle subscriptions.

```
Observable.combineLatest(
    RxTextView.textChangeEvents(mEmailView),
    RxTextView.textChangeEvents(mPasswordView),
    (emailChange, passwordChange) -> {
        boolean emailOK = emailChange.text().length() >= 3;
        boolean passOK = passwordChange.text().length() >= 3;

        return emailOK && passOK;
    })
    .compose(bindToLifecycle())
    .subscribe(aBoolean -> mSignIn.setEnabled(aBoolean));
```

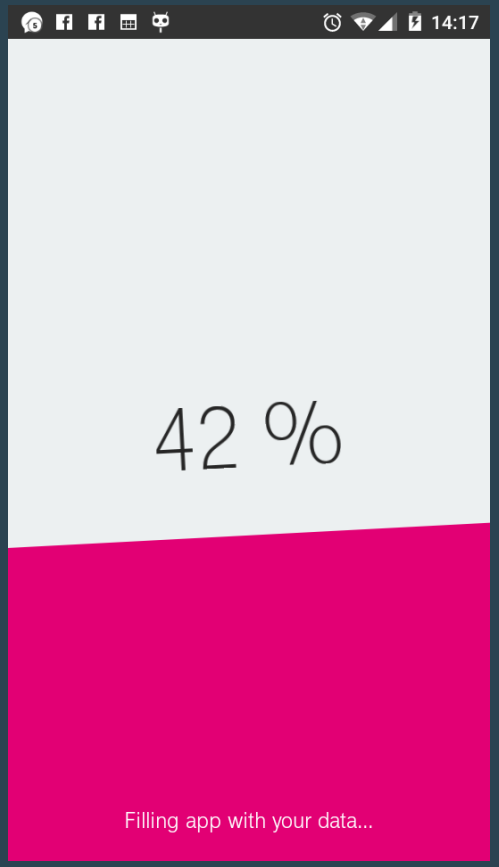
Advanced Scenario

- Enable/Disable Sign in button

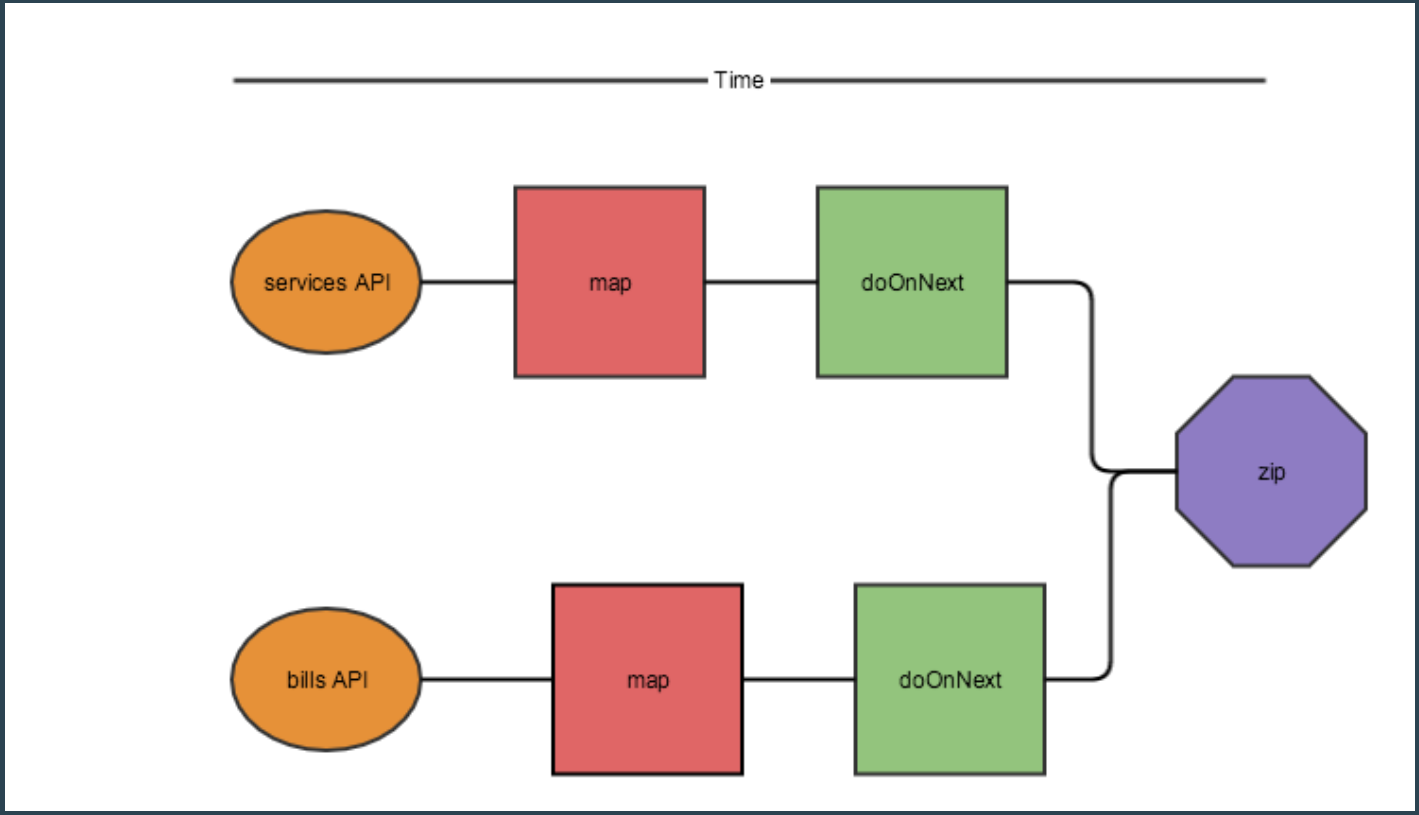
```
Observable.combineLatest(
    RxTextView.textChangeEvents(mEmailView),
    RxTextView.textChangeEvents(mPasswordView),
    (emailChange, passwordChange) -> {
        boolean emailOK = emailChange.text().length() >= 3;
        boolean passOK = passwordChange.text().length() >= 3;

        return emailOK && passOK;
    })
    .compose(bindToLifecycle())
    .subscribe(aBoolean -> mSignIn.setEnabled(aBoolean));
```

Download Data



Advanced Scenario



Advanced Scenario

- Process services first.

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));
```

Advanced Scenario

- Download data with Retrofit

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));
```

```
@GET("/services")
Observable<ServiceResponse> services();
```

Advanced Scenario

- Apply schedulers

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));
```

```
public interface RxComposer {
<T> Observable.Transformer<T, T> applyDefaultSchedulers();
}
```

Advanced Scenario

- Apply schedulers

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));


public <T> Observable.Transformer<T, T>
    applyDefaultSchedulers() {
        return obs-> obs.subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }
}
```

Advanced Scenario

- Apply schedulers

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));


public <T> Observable.Transformer<T, T>
    applyDefaultSchedulers() {
        return obs-> obs.subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }
}
```

doInBackground

Advanced Scenario

- Apply schedulers

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));


public <T> Observable.Transformer<T, T>
    applyDefaultSchedulers() {
        return obs-> obs.subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread());
    }
}
```

runOnUiThread

Advanced Scenario

- Map response to domain model

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));
```

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

Advanced Scenario

- Save data to database

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));
```

Advanced Scenario

- Do things parallel

```
Observable<Services> saveServices = apiManager.services()
    .compose(rxUtils.applySchedulers())
    .map(serviceResponse -> mapServices(serviceResponse))
    .doOnNext(services -> saveServices(services));

Observable<Bills> saveBills = apiManager.bills()
    .compose(rxUtils.applySchedulers())
    .map(billResponse -> mapBills(billResponse))
    .doOnNext(bills -> saveBills(bills));
```

Advanced Scenario

- Do things parallel

```
Observable.zip(saveServices, saveBills,
    (s, b) -> new LoadingResult(s, b))
      .compose(rxComposer.applySchedulers())
      .subscribe(loadingResult -> log(loadingResult));
```

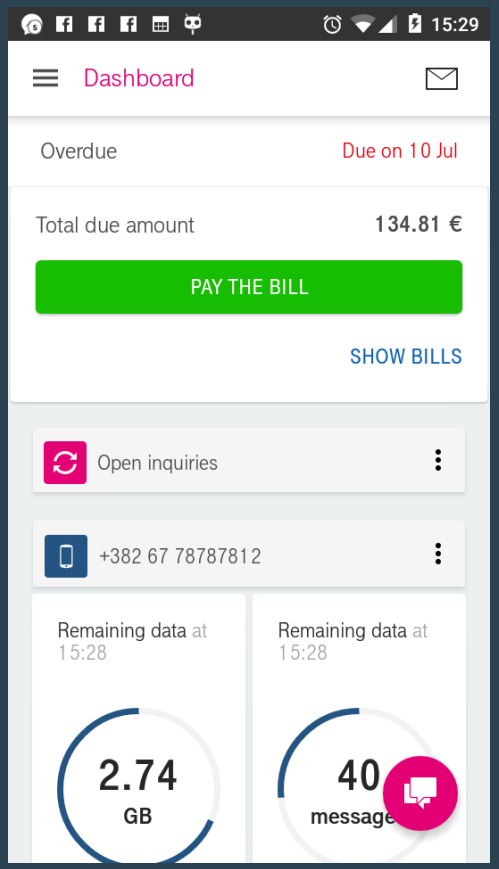
Advanced Scenario

- Do things parallel

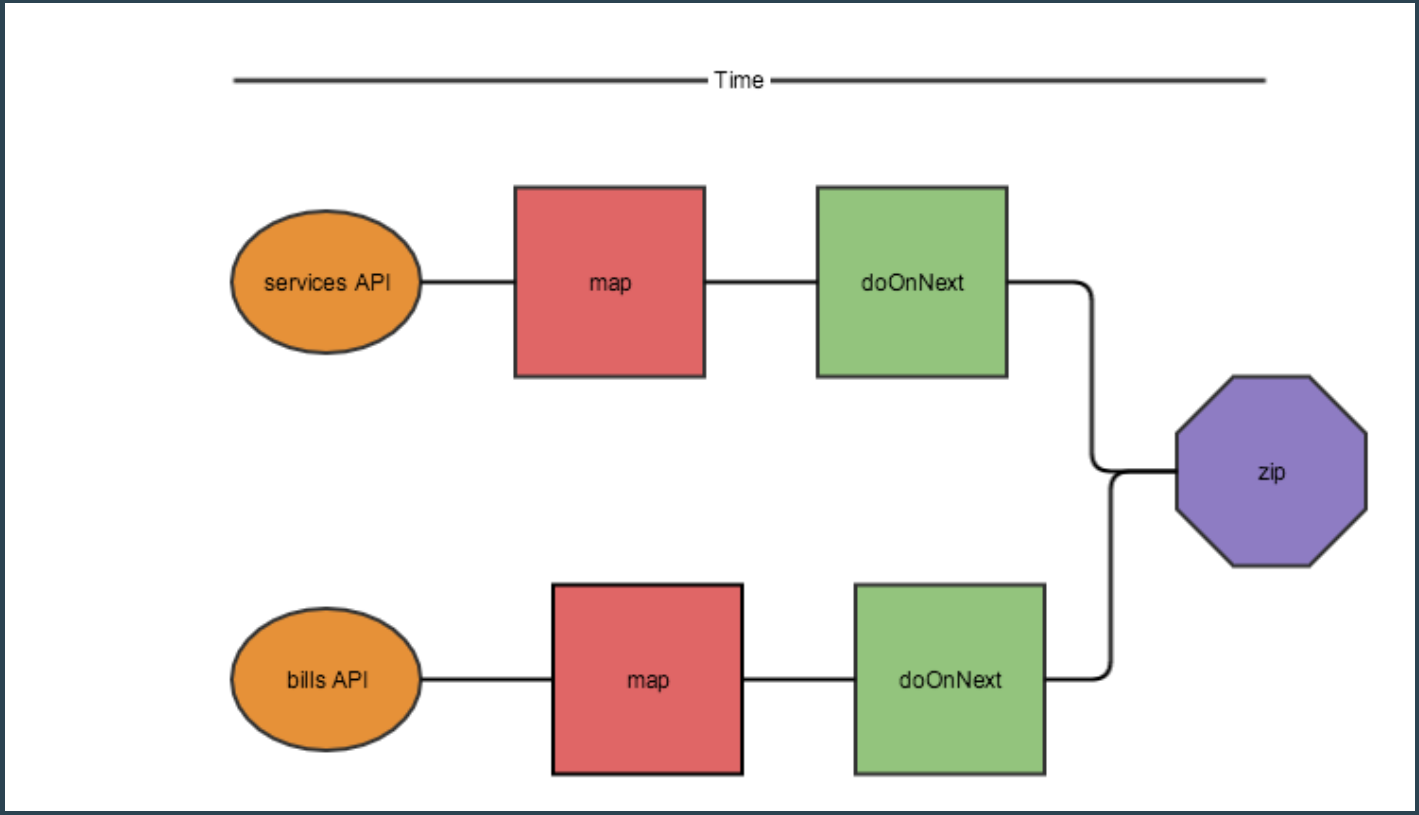
```
Observable.zip(saveServices, saveBills,
    (s, b) -> new LoadingResult(s, b))
      .compose(rxComposer.applySchedulers())
      .subscribe(loadingResult -> log(loadingResult));


static class LoadingResult{
    public LoadingResult(Services service, Bills bill)
    {…}
}
```

Dashboard


15:29
Dashboard
Overdue
Due on 10 Jul
Total due amount
134.81 €
PAY THE BILL
SHOW BILLS
Open inquiries
+382 67 78787812
Remaining data at 15:28
2.74
GB
Remaining data at 15:28
40
message

Advanced Scenario



Testing

- What to test?

Testing

- What to test?
  - Observables: composition of the various operators

Testing

- What to test?
  - Observables: composition of the various operators
  - How the rest of app behaves while triggered by a subscription

ToBlocking

```
ResultToCheck res = myObservable.toBlocking().first();
```

Converts the observable to BlockingObservable.
The code waits synchronously for onCompleted.

TestSubscriber

```
Observable<Data> obs = getMyObservable();
TestSubscriber<Data> testSubscriber = new
TestSubscriber<>();
obs.subscribe(testSubscriber);

testSubscriber.assertNoErrors();
List<Data> datas = testSubscriber.getOnNextEvents();
```

Official way.

Comes with bunch of helper methods.

RxJava

Thanks for your attention!

Contact us!

Richard Radics

Supercharge

Android Developer

richard.radics@supercharge.io // www.supercharge.io

Questions?

Supercharge

References

https://github.com/ReactiveX/RxJava

https://medium.com/swlh/party-tricks-with-rxjava-rxandroid-retrolambda-1b06ed7cd29c

http://fernandocejas.com/2015/07/18/architecting-android-the-evolution/

http://blog.danlew.net/2014/09/15/grokking-rxjava-part-1/

http://blog.danlew.net/2015/07/23/deferring-observable-code-until-subscription-in-rxjava

http://futurice.com/blog/top-7-tips-for-rxjava-on-android

http://fernandocejas.com/2015/01/11/rxjava-observable-tranformation-concatmap-vs-flatmap/

http://alexismas.com/blog/2015/05/20/unit-testing-rxjava/

https://medium.com/ribot-labs/unit-testing-rxjava-6e9540d4a329

http://akarnokd.blogspot.hu/