
RxFy All The Things!
No pain and easy refactoring

Benjamin Augustin

Android Software Craftsman

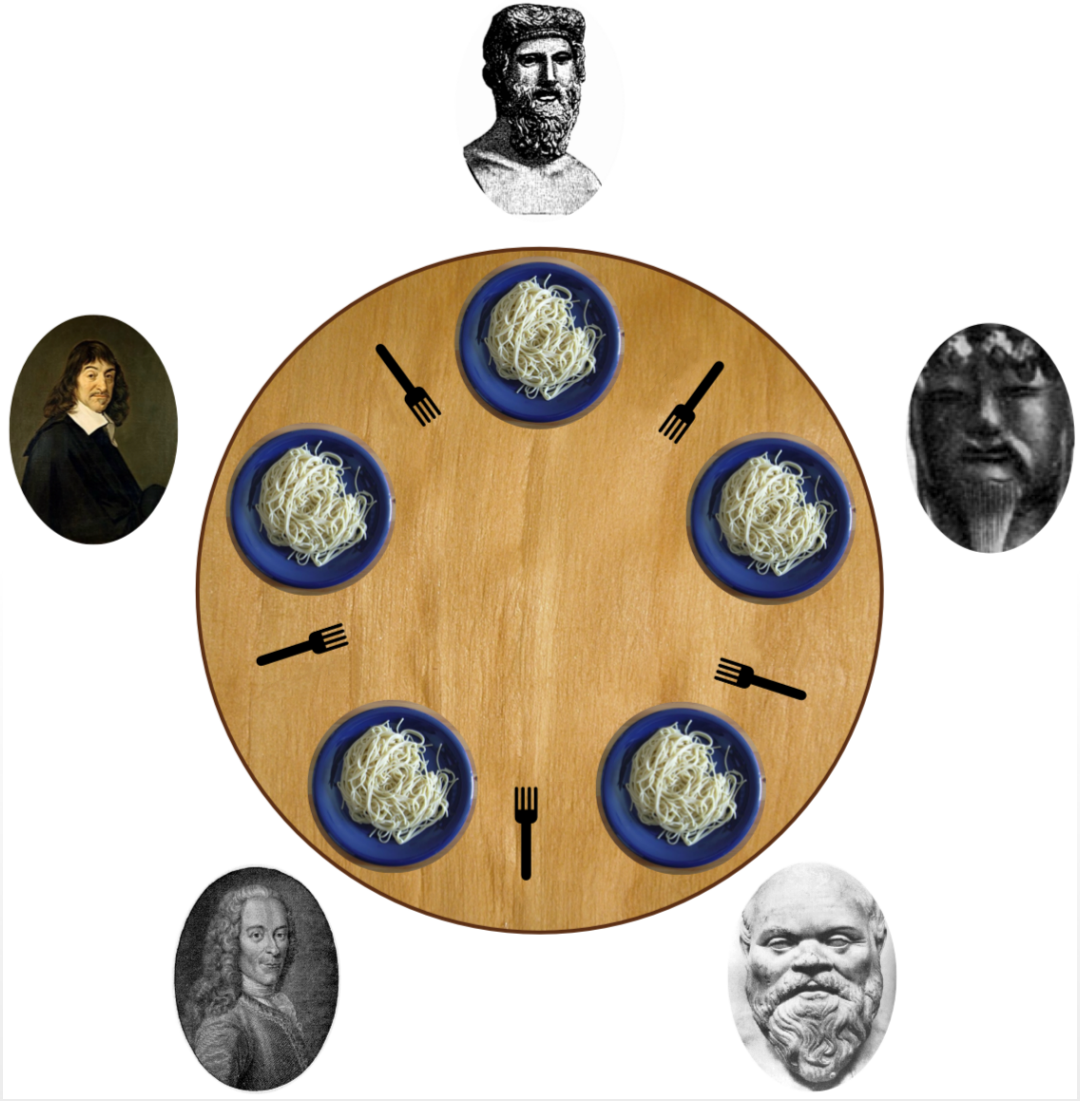
@Dorvaryn

+BenjaminAugustin-Dorvaryn

Dorvaryn


RXFY
ALL THE THINGS !

Why ?



Development is hell

So what is RxJava ?

What is an Observable ?

	Single	Multiple
Sync	`T getData()`	`Iterable<T> getData()`
Async	`Future<T> getData()`	`Observable<T> getData`

What is an Observable ?

What is an Observer ?

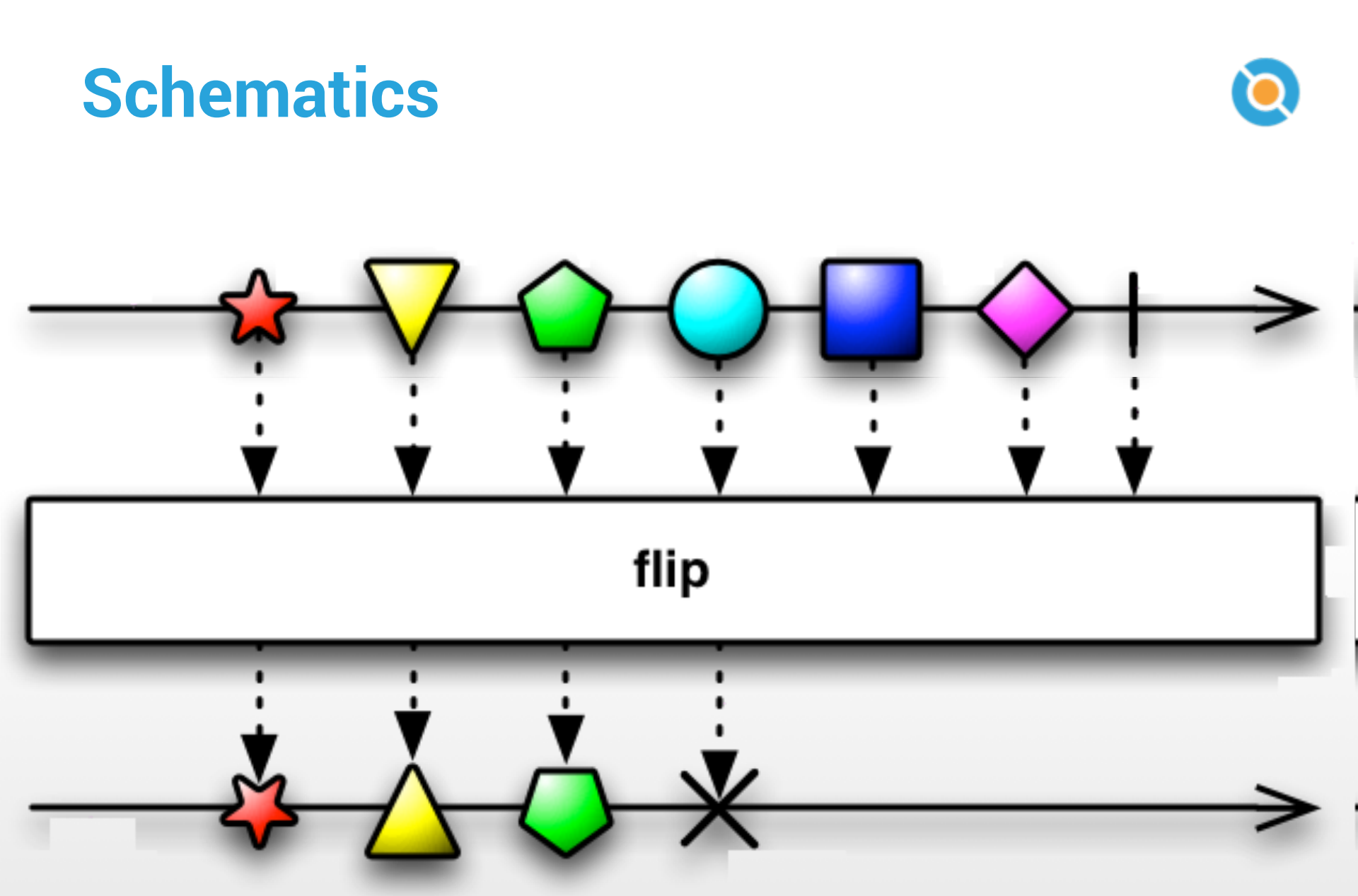
Resource

RxJava wiki

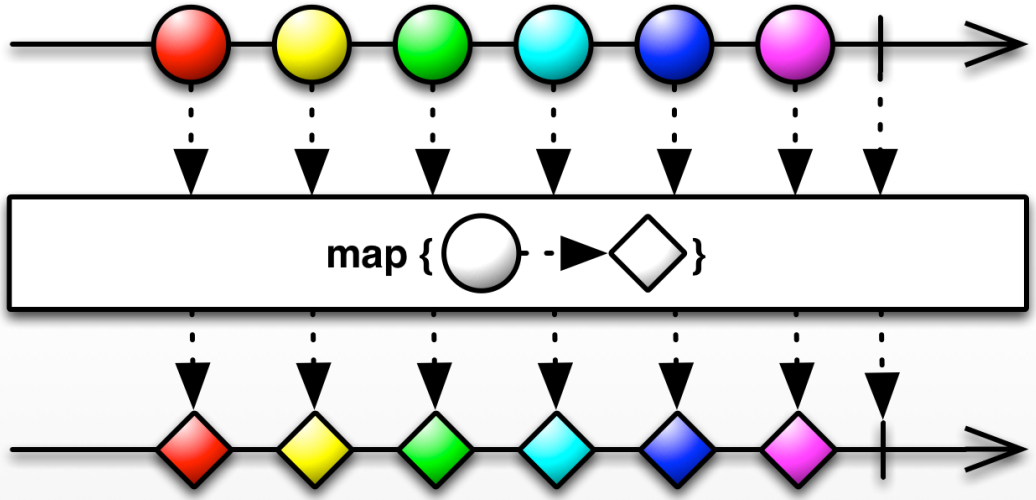
all the help you need

https://github.com/ReactiveX/RxJava/wiki

Schematics

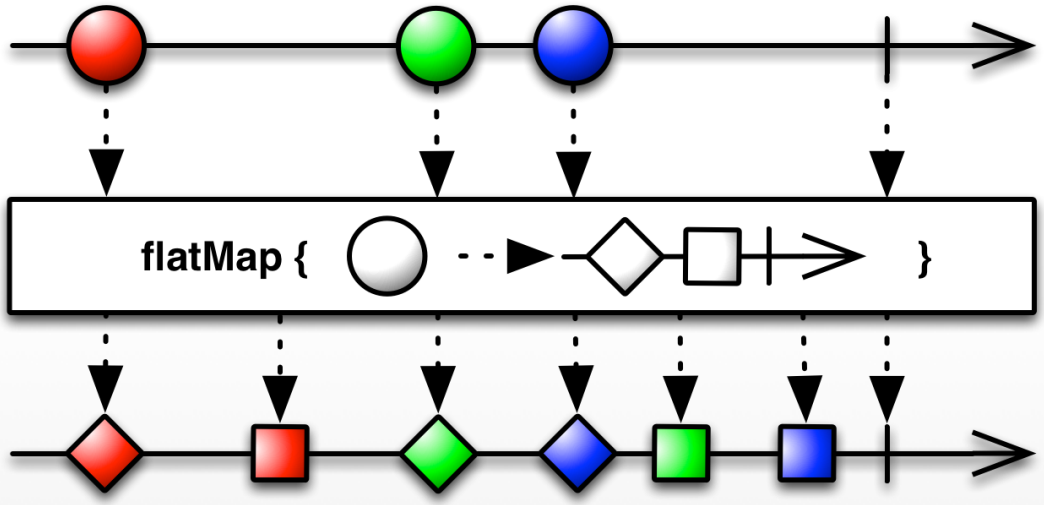


map



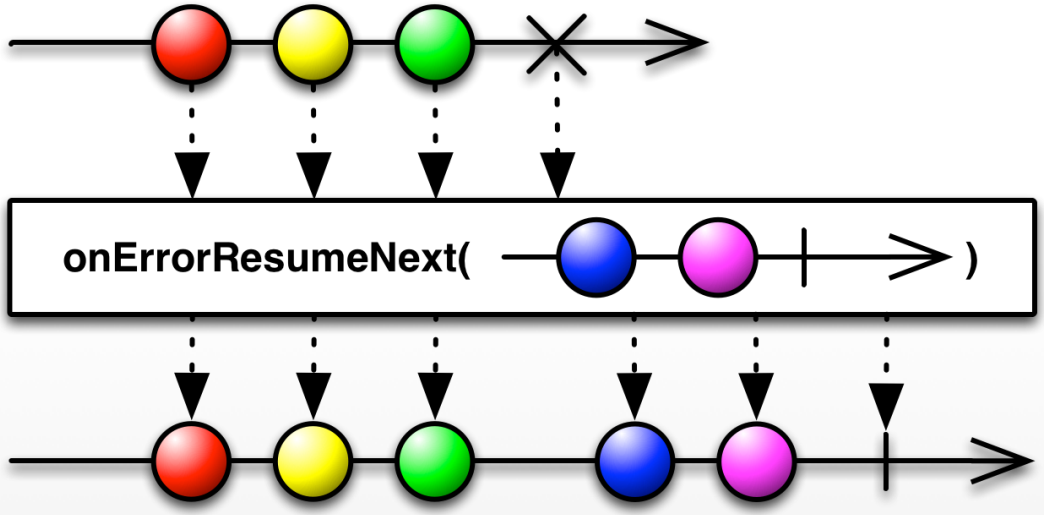
- Function
- Transformation
- For each element

flatMap



- Function
- Composition
- For each element

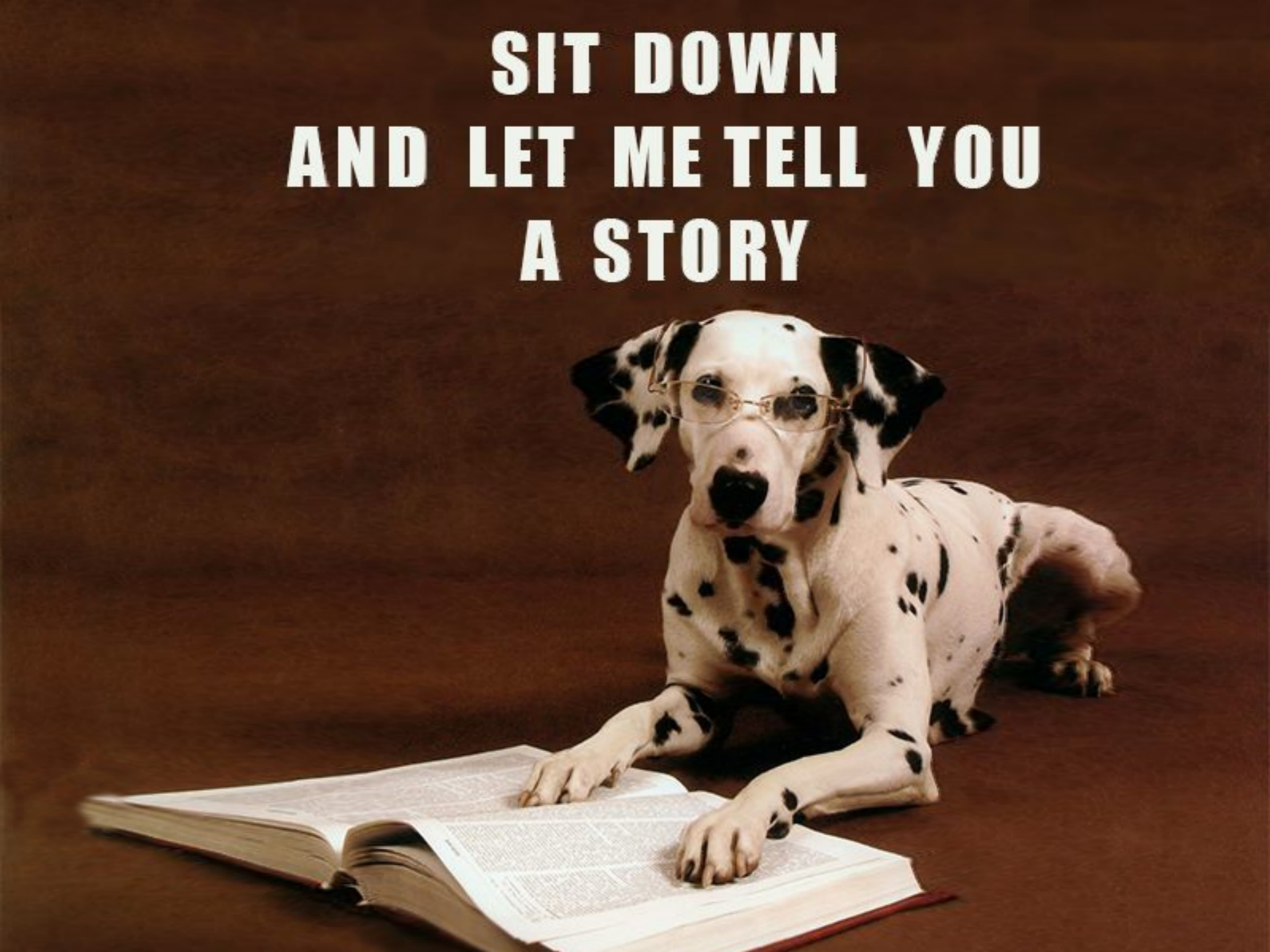
onErrorResumeNext



- Function
- Recovery
- On Failure

Async control

- `subscribeOn(Scheduler)` :
  - Execute observable on Scheduler
- `observeOn(Scheduler)` :
  - Deliver results to observer on Scheduler


SIT DOWN
AND LET ME TELL YOU
A STORY

All is starting well

- addToBasket
  - Take a barcode
  - Return a basket

Easy enough right ?

```
private class AddToBasket extends AsyncTask<Barcode, Void, Basket> {

        private final API api;
        private final Basket currentBasket;
        private final BasketViewUpdater updater;

        AddToBasket(API api, Basket currentBasket, BasketViewUpdater updater) {
            this.api = api;
            this.currentBasket = currentBasket;
            this.updater = updater;
        }

        @Override
        protected Basket doInBackground(Barcode... barcodes) {
            return api.addToBasket(barcodes[0]);
        }

        @Override
        protected void onPostExecute(Basket response) {
            super.onPostExecute(response);
            updater.updateWith(response);
        }
}
```

After 4 months...

```
private class Ad

        private
        private
        private

        AddToBas
            this
            this
            this
        }

        @Overrid
        protecte
            retu
        }

        @Overrid
        protecte
            supe
            new
        }
    }

    private clas

        private
        private

        CheckFor
            this
            this
        }

        @Overrid
        protecte
            Asyn
            if(r

            }

            Resp
            if (

            }
            retu
        }

        @Overrid
        protected void onPostExecute(AsyncResult<Diff> basket) {
            super.onPostExecute(basket);
            new ApplyDiff(currentBasket, updater).execute(basket);
        }
    }

    private class ApplyDiff extends AsyncTask<AsyncResult<Diff>, Void, AsyncResult<Basket>> {

        private final Basket currentBasket;
        private final BasketViewUpdater updater;
```

So how to RxFy it ?

Rx Style

```
class BasketObserver implements Observer<Basket> {
    public onNext(Basket basket) {
        updateUI(basket);
    }

    public onError(Throwable error) {
        displayDialogFor(error);
    }

    public onComplete() {
    }
}
```

Rx Style

Definition:

```
public Observable<Basket> addToBasket(Barcode code);
```

Usage:

```
addToBasket(code)
   .subscribe(new BasketObserver());
```

HTTP Standard ?

Refactoring

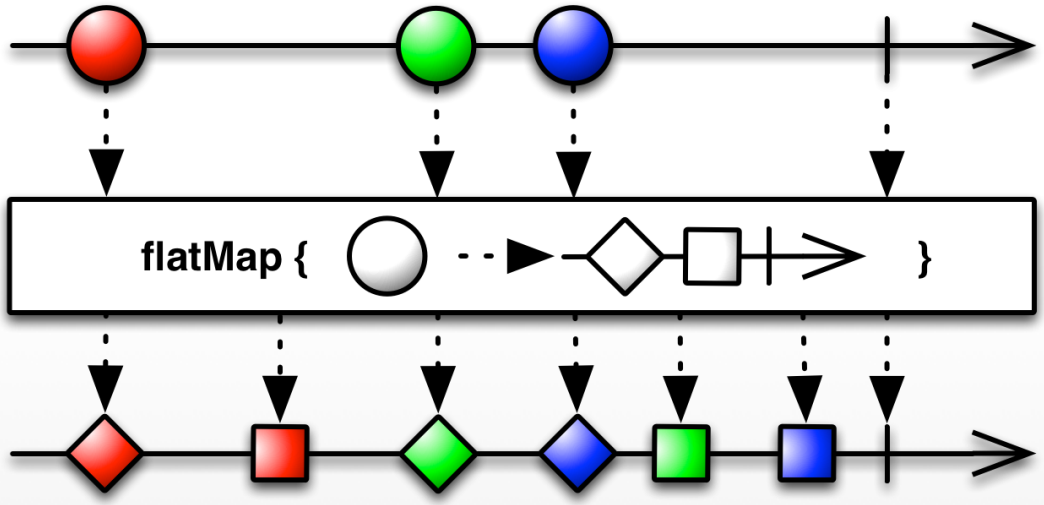
Definition:

```
public Observable<Basket> addToBasket(Barcode code) {
     return addToBasketReq(code).flatmap(checkAPISuccess());
}
private Observable<Response<Basket>> addToBasketReq(Barcode code);
```

Usage:

```
addToBasket(code)
    .subscribe(new BasketObserver());
```

flatMap



- Function
- Composition
- For each element

Function

```
public Func1<Response<Basket>, Observable<Basket>> checkApiSuccess() {
        return new Func1<Response<Basket>, Observable<Basket>>() {

            @Override
            public Observable<Basket> call(Response<Basket> response) {
                if (response.isSuccess()) {
                    return Observable.from(response.getBody());
                }
                return Observable.error(new CustomError(header));
            }

        };
}
```

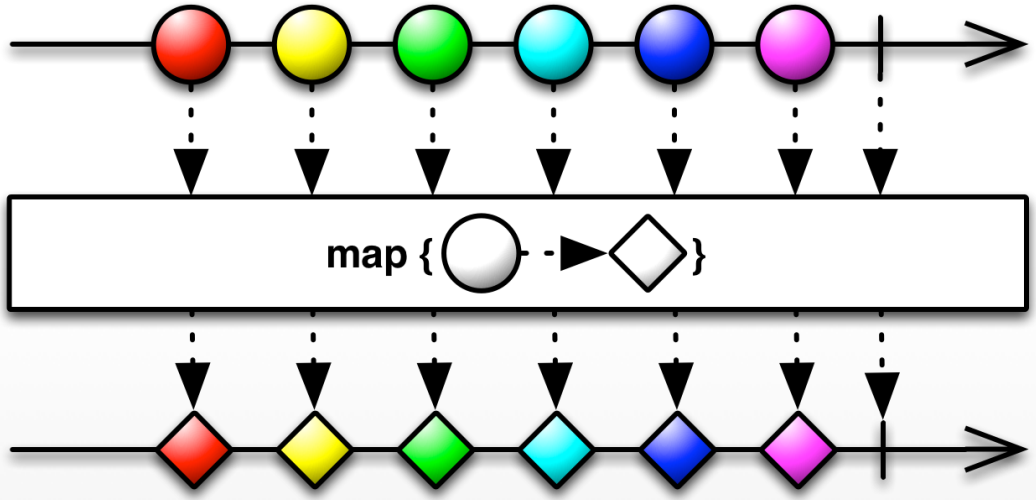
Stateless ?

Refactoring

Definition:

```
public Observable<Basket> addToBasket(Barcode code) {
     return addToBasketReq(code)
                    .flatmap(checkAPISuccess())
                    .map(applyDiffTo(cachedBasket));
}
```

map



- Function
- Transformation
- For each element

Function

```
public Func1<Diff, Basket> applyDiffTo(final Basket basket) {
        return new Func1<Diff, Basket>() {

            @Override
            public Basket call(Diff diff) {
                return basket.apply(diff);
            }

        };
}
```

Logic server side ?

Refactoring

Definition:

```
public Observable<Basket> addToBasket(Barcode code) {
     return addToBasketReq(code)
                    .flatmap(checkAPISuccess())
                    .flatmap(addCustomerIfNeeded())
                    .map(applyDiffTo(cachedBasket));
}
```

Function

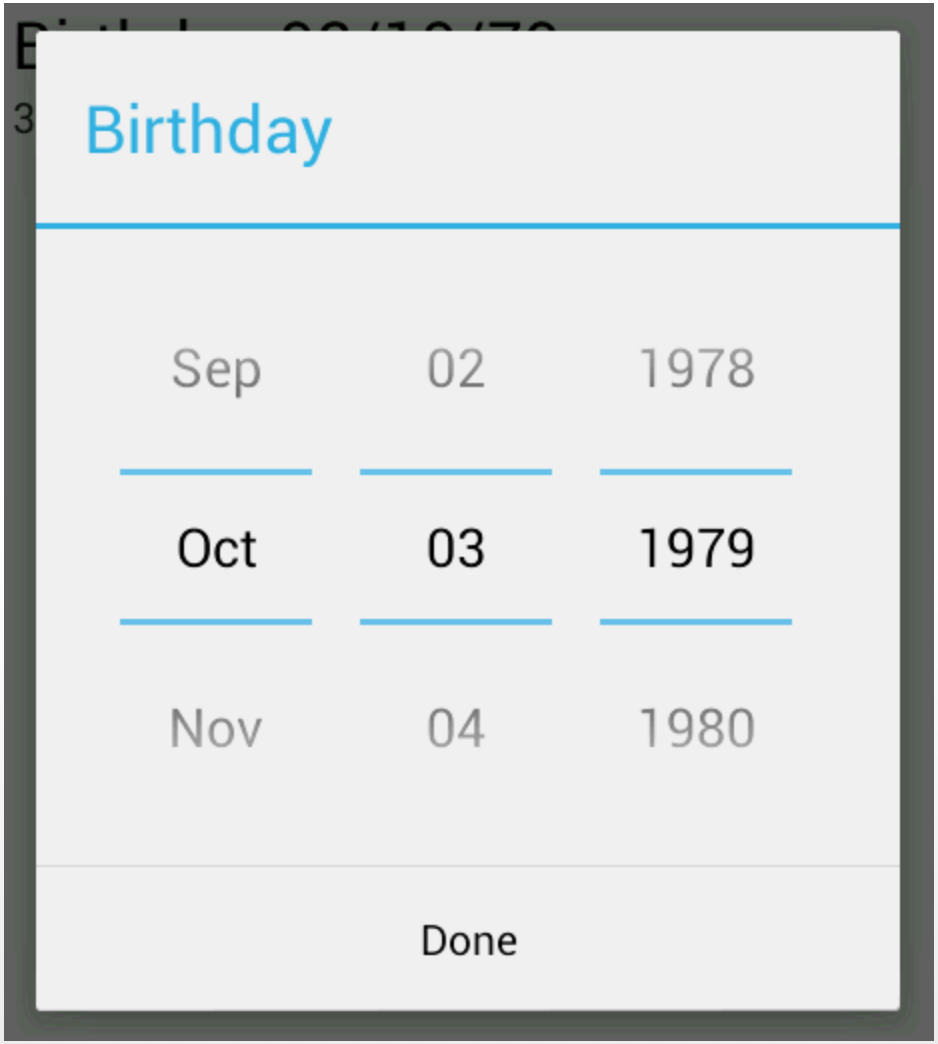
```
private Func1<Diff, Observable<Diff>> addCustomerIfNeeded() {
        return new Func1<Diff, Observable<Diff>>() {

            @Override
            public Observable<Diff> call(Diff diff) {
                if (diff.requireAddCustomer()) {
                    return addCustomer(diff.getCustomerId());
                }
                return Observable.from(diff);
            }

        };
    }
```

Oh what about the law ?

UI in network flow ?

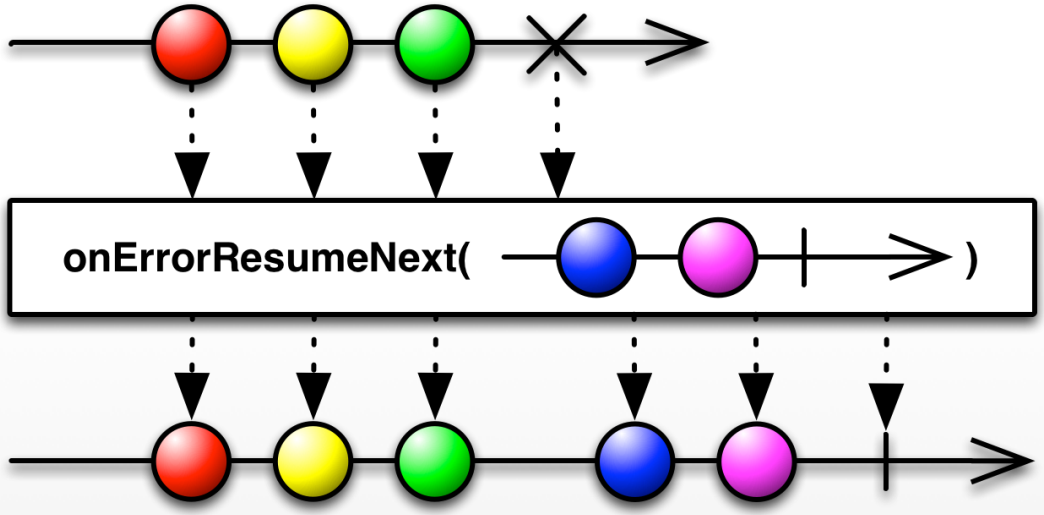


Refactoring

Definition:

```
public Observable<Basket> addToBasket(Barcode code) {
     return addToBasket(code, Data.emptyData);
}
public Observable<Basket> addToBasket(Barcode code, Data data) {
     return addToBasketReq(code, data)
               .flatmap(checkAPISuccess())
               .flatmap(addCustomerIfNeeded())
               .map(applyDiffTo(cachedBasket))
               .onErrorResumeNext(checkForError(continuation(code)));
}
```

onErrorResumeNext



- Function
- Recovery
- On Failure

Function

```
public Func1<Throwable, Observable<Basket>> checkForError(final Func1<Data,
Observable<Basket>> continuation) {
  return new Func1<Throwable, Observable<Basket>>() {


    @Override
    public Observable<Basket> call(Throwable error) {
        if (error instanceof BusinessRuleExeption) {
            return Observable.error(new NeedMoreDataException(continuation));
        }
        return Observable.error(error);
    }


  };
}
```

Function

```
public Func1<Data, Observable<Basket>> continuation(final Barcode
barcode) {
     return new Func1<Data, Observable<Basket>> {

          @Override
           public Observable<Basket> call (Data data){
               return addToBasket(barcode, data);
           }

     }
}
```

Observer

```
class BasketObserver implements Observer<Basket> {
    public onNext(Basket basket) {
        updateUI(basket);
    }

    public onError(Throwable error) {
        displayDialogFor(error).flatmap(withContinuationFrom(error));
    }

    public onComplete() {
    }
}
```

Function

```
private Observable<Data> displayDialogFor(Throwable error);


private Func1<Data, Observable<Basket>> withContinuationFrom(NeedMoreData
error) {
        return new Func1<Data, Observable<T>>() {

            @Override
            public Observable<Basket> call(final Data data) {
                return error.getContinuation().call(data);
            }

        };
}
```

Dialog is reactive

```
public class ReactiveDialog<T> extends DialogFragment {
    public Observable<T> show(final FragmentManager manager) {

        return Observable.create(new Observable.OnSubscribe<T>() {
            @Override
            public void call(rx.Subscriber<? super T> subscriber) {
                UUID key = subscriberVault.store(subscriber);
                getArguments().putSerializable(REACTIVE_DIALOG_KEY, key);
                show(manager, getClass().getSimpleName());
            }
        });

    }
}
```

Reactive Dialogs ?!

Recap

Definition:

```
public Observable<Basket> addToBasket(Barcode code, Data data) {
     return addToBasketReq(code, data)
                .flatmap(checkAPISuccess())
                .flatmap(addCustomerIfNeeded())
                .map(applyDiffTo(cachedBasket))
                .onErrorResumeNext(checkForError(continuation(code, data)));
}
```

Resource

RxAndroid

collection of tools for RxJava on Android

https://github.com/ReactiveX/RxAndroid

Resource

RxJava workshop

a simple workshop to start

http://goo.gl/el3hMc

We’re hiring !

join@novoda.com

Questions ?

Benjamin Augustin

Android Software Craftsman

@Dorvaryn

+BenjaminAugustin-Dorvaryn

Dorvaryn