RxJava

Горев Артём
BSC Владимир

Асинхронные операции

Асинхронные операции

1. Сложная обработка ошибок
2. Много явных и неявных состояний
3. Callback-hell
4. Трудности в поддержке и рефакторинге
5. Утечки памяти

Reactive E**x**tension

Единый подход

- Rx.NET
- RxJava
- RxClojure
- RxJS
- RxScala
- RxSwift
- RxCpp
- …

Observable

- Может «выпустить» ноль, один или несколько элементов
- Имеет два терминальных состояния

Каноническая реализация Observable

```
Observable.create(subscriber -> {
    try {
        subscriber.onNext("Hello World");
        subscriber.onCompleted();
    } catch (Exception e) {
        subscriber.onError(e);
    }
});
```

Observable + Observer

```
Observable.create(new Observable.OnSubscribe<String>() {
    @Override
    public void call(Subscriber<? super String> subscriber) {
        try {
            subscriber.onNext("Hello World");
            subscriber.onCompleted();
        } catch (Exception e) {
            subscriber.onError(e);
        }
    }
}).subscribe(new Observer<String>() {
    @Override
    public void onCompleted() {
        log("Complete");
    }

    @Override
    public void onError(Throwable e) {
        log(e);

    }

    @Override
    public void onNext(String next) {
        log(next);

    }
});
```

(un)Subscription

```
Subscription subscription =
        Observable.create(new Observable.OnSubscribe<String>() {...})
                .subscribe(new Observer<String>() {...});

subscription.unsubscribe();
```

Операторы

Фильтрующие операторы

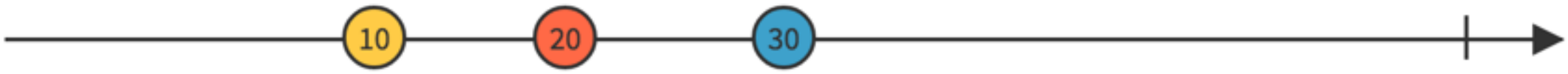

```
filter(x => x > 10)
```

Фильтрующие операторы

```
public void filterEditText() {
    RxTextview.textChanges(searchEditText)
            .filter(text -> text.length() > 3);
}
```

Трансформирующие операторы

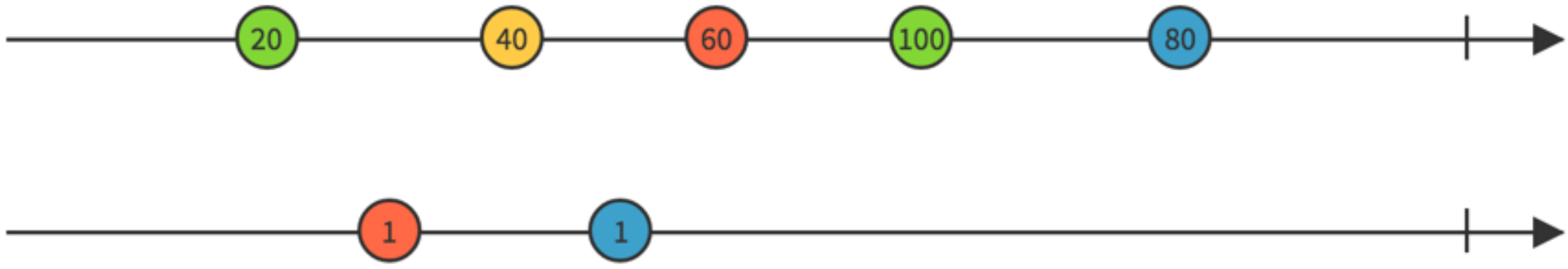
map(x => 10 * x)



Трансформирующие операторы

```
public void filterEditText() {
    RxTextview.textChanges(searchEditText)
            .map(text -> text.toString().trim())
            .filter(text -> text.length() != 0);
}
```

Комбинирующие операции



merge

Комбинирующие операции

```
Observable.combineLatest(
        RxTextView.textChanges(loginEditText),
        RxTextView.textChanges(passwordEditText),
        (login, password) -> login.length() > 0 && password.length() > 0)
        .subscribe(loginButton::setEnabled);
```

Комбинирующие операции

```
Observable.zip(
        service.getUserPhoto(id),
        service.getPhotoMetadata(id),
        (photo, metadata) -> createPhotoWithData(photo, metadata))
        .subscribe(photoWithData -> showPhoto(photoWithData));
```

Шедулинг

subscribeOn(Scheduler)

Указывает на каком потоке нужно выполнить начало подписки

observeOn(Scheduler)

Указывает на каком потоке нужно наблюдать за данными

Шедулинг

```
retrofitService.getImage(url)
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(bitmap -> myImageView.setImageBitmap(bitmap));
```

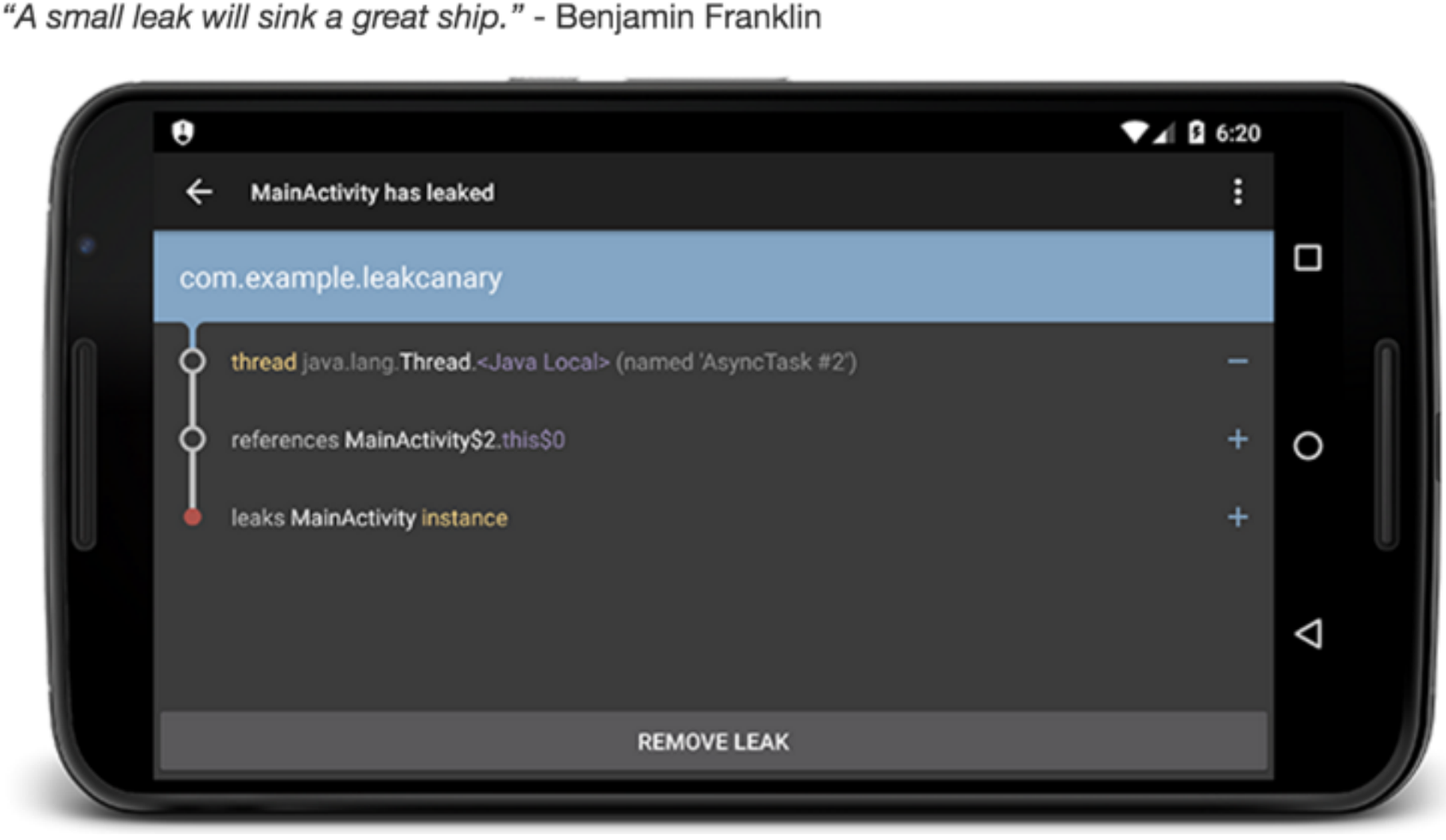
Reactive Android

- https://github.com/ReactiveX/RxAndroid
- https://github.com/orfjackal/retrolambda
- https://github.com/ReactiveX/RxJava
- https://github.com/JakeWharton/RxBinding

- http://square.github.io/retrofit/
- https://github.com/ragnraok/RxCamera
- https://github.com/pushtorefresh/storio
- https://github.com/VictorAlbertos/RxCache
- https://github.com/VictorAlbertos/RxGcm
- https://github.com/Mauin/RxFingerprint
- https://github.com/aaronhe42/RxGoogleMapsBinding

Полезняшка

https://github.com/square/leakcanary



Спасибо за внимание