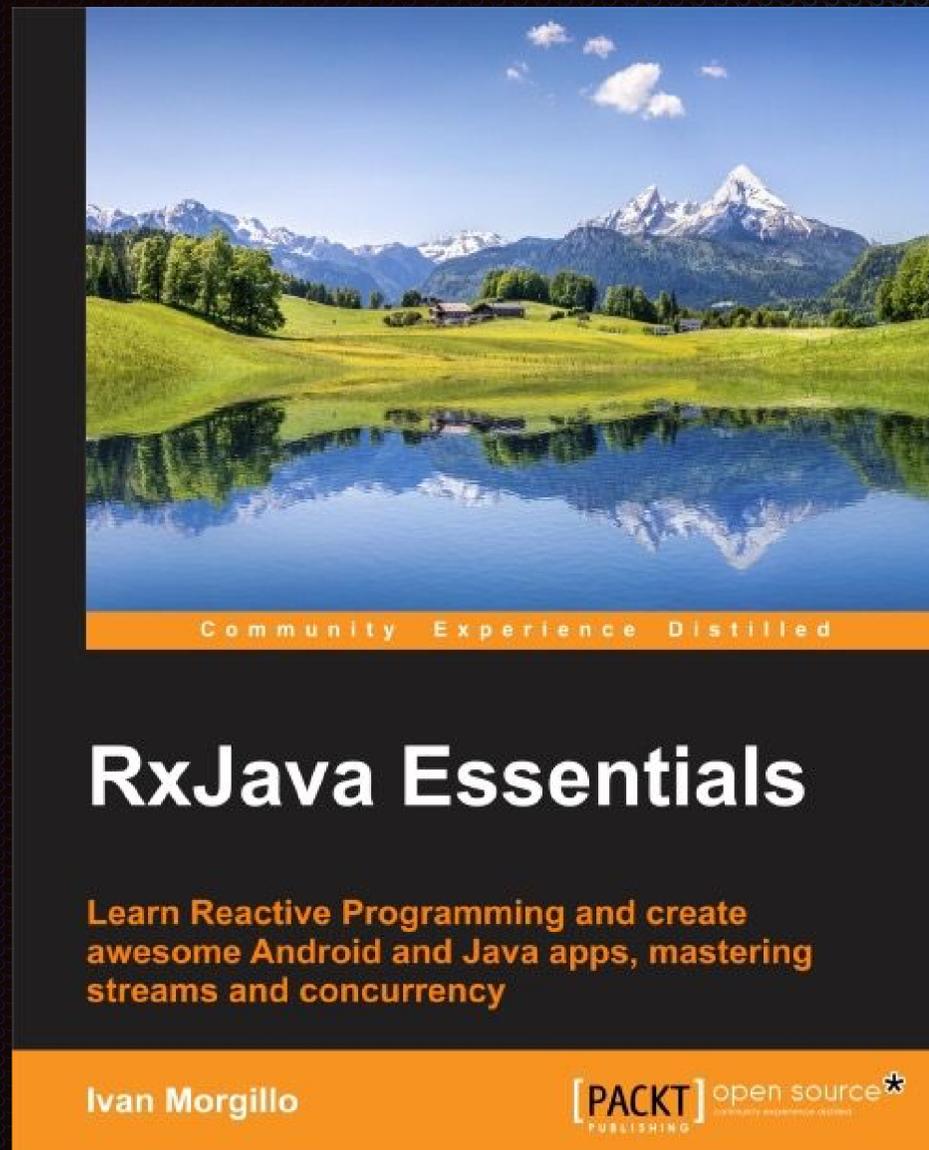
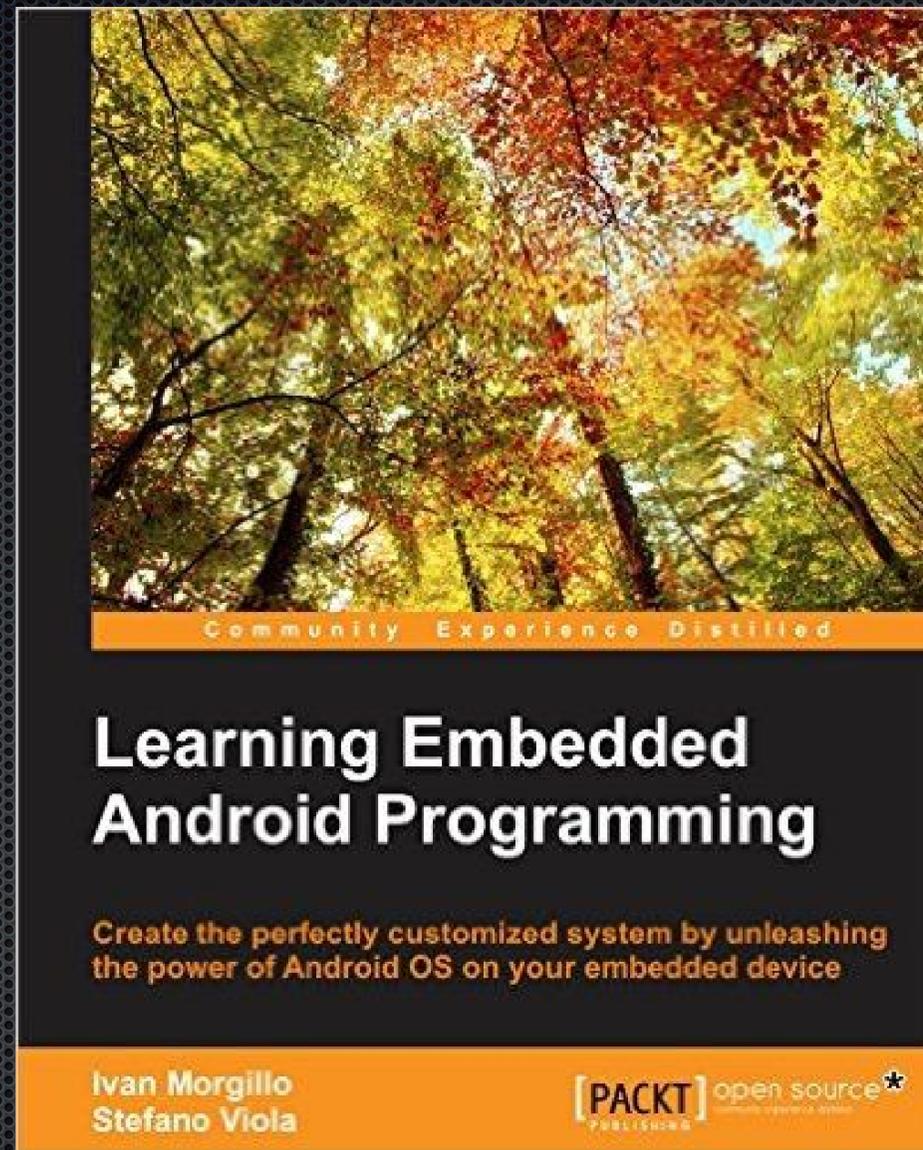


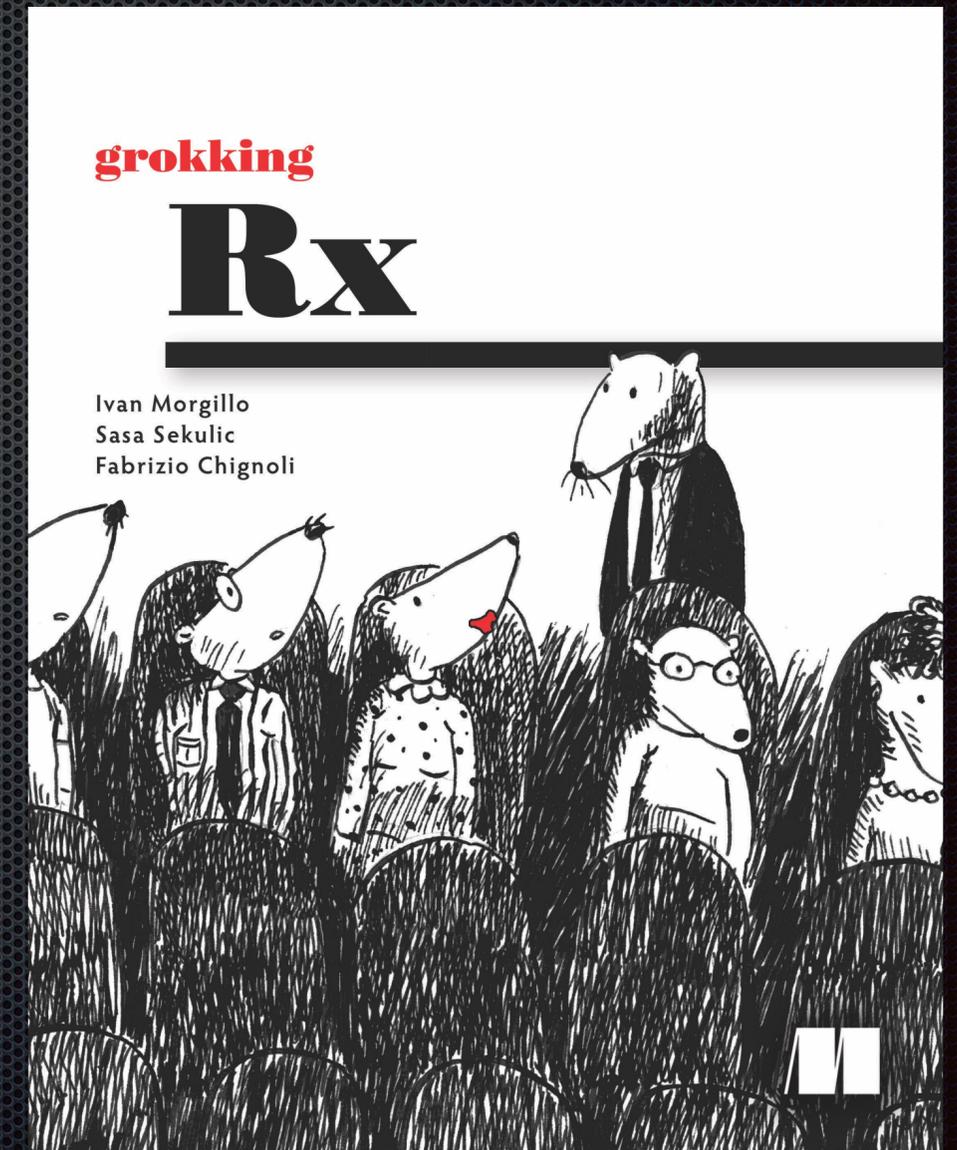
Ivan Morgillo



[http://bit.ly/
rxjava-essentials](http://bit.ly/rxjava-essentials)



[http://bit.ly/
learning-embedded-android-programming](http://bit.ly/learning-embedded-android-programming)

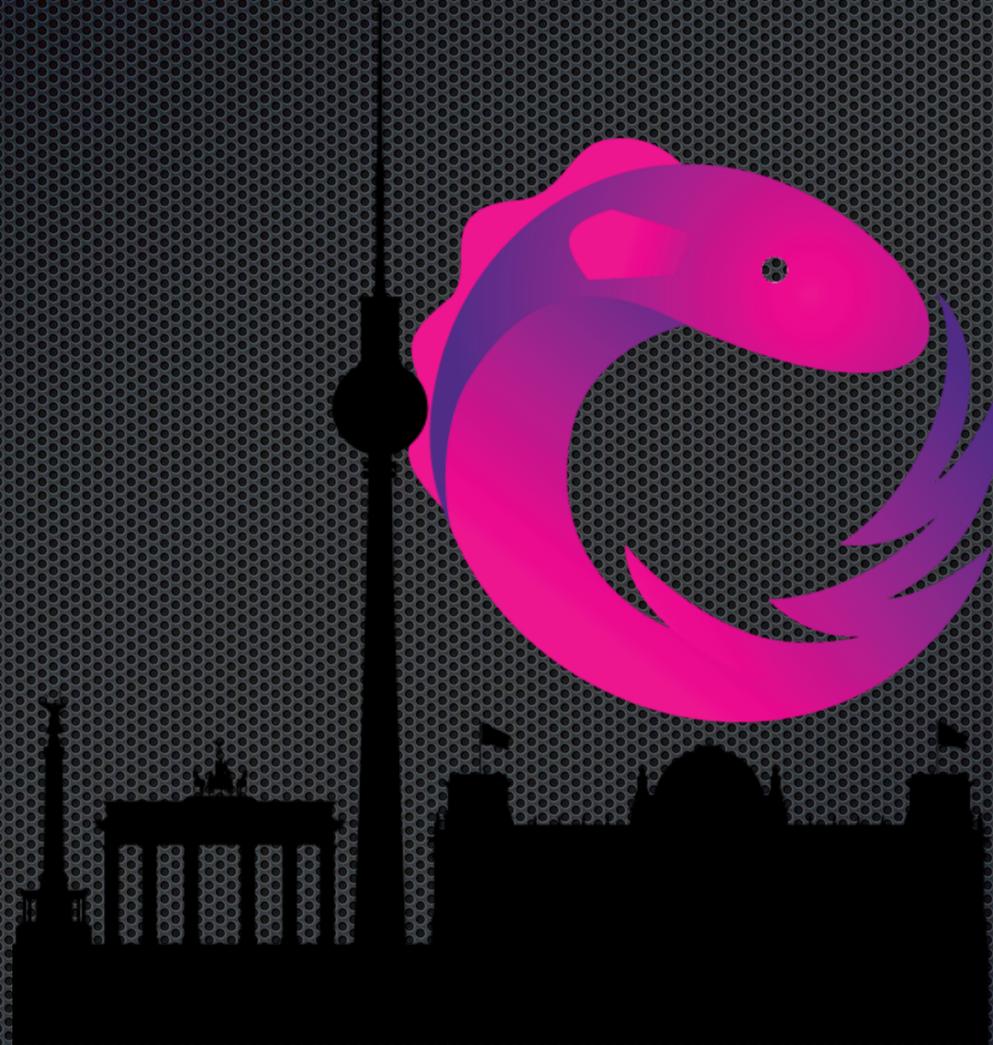


[http://bit.ly/
grokking-rx](http://bit.ly/grokking-rx)

Ivan Morgillo



Android Reactive Programming with RxJava



Reactive Programming



Erik Meijer - Rx .Net



Ben Christensen - RxJava



Dávid Karnok

Reactive Programming

- 📌 Data in charge
- 📌 Observer Pattern
- 📌 Push vs Pull

Observer Pattern

 Notifications

 Automation

 Cause-Effect

Rx Observer Pattern

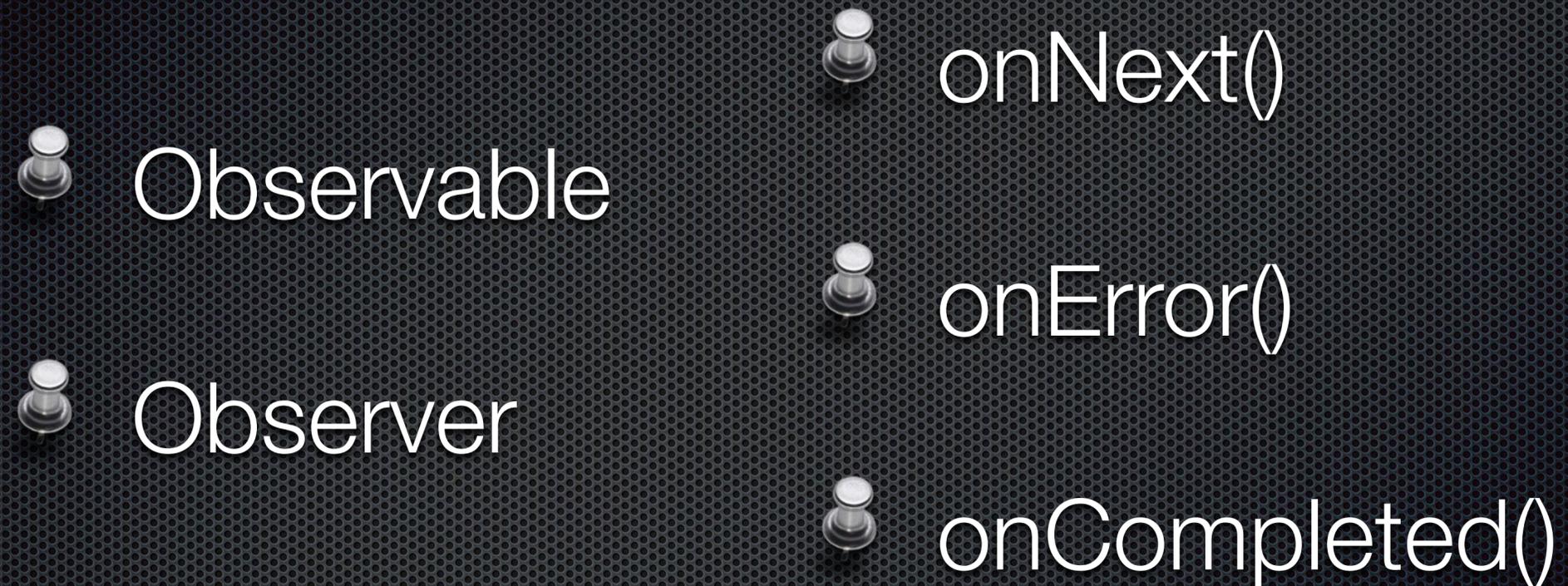
 Observable

 Observer

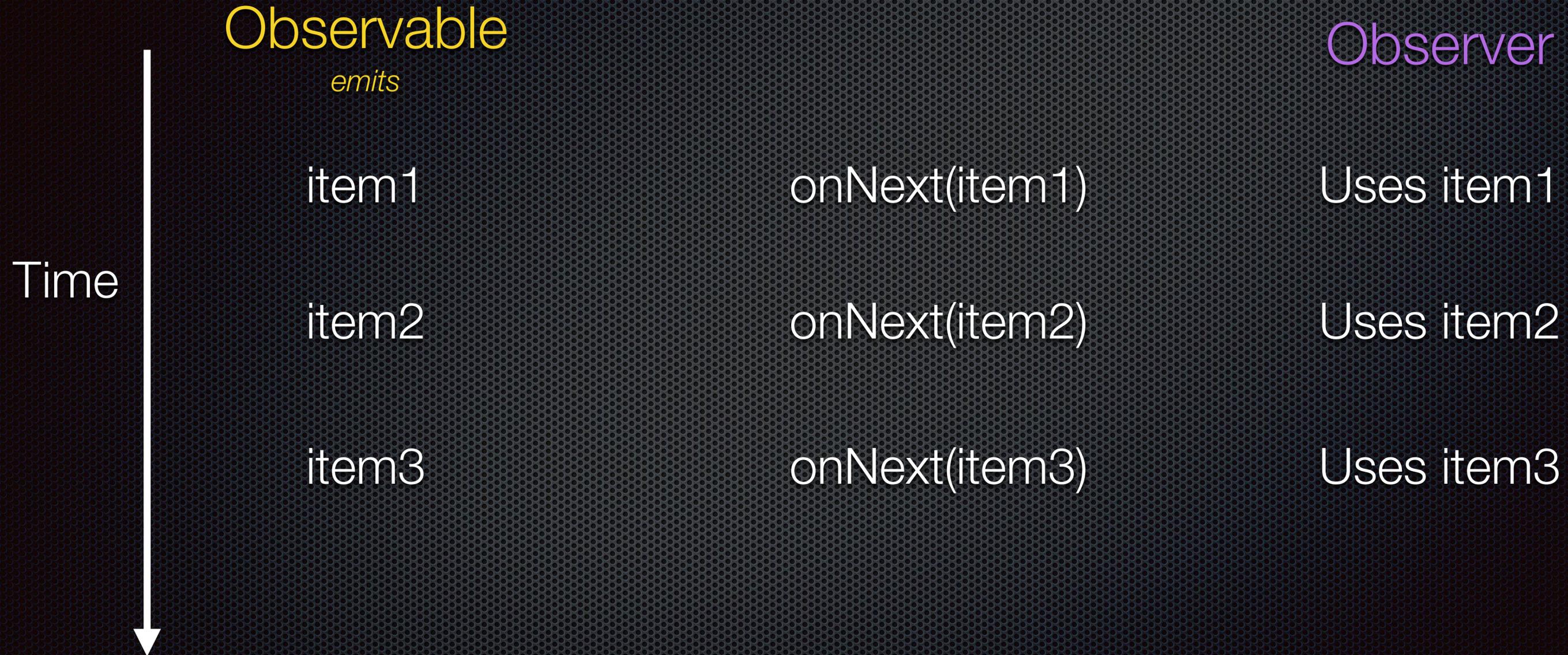
 Subscription

 Subject

Rx Observer Pattern

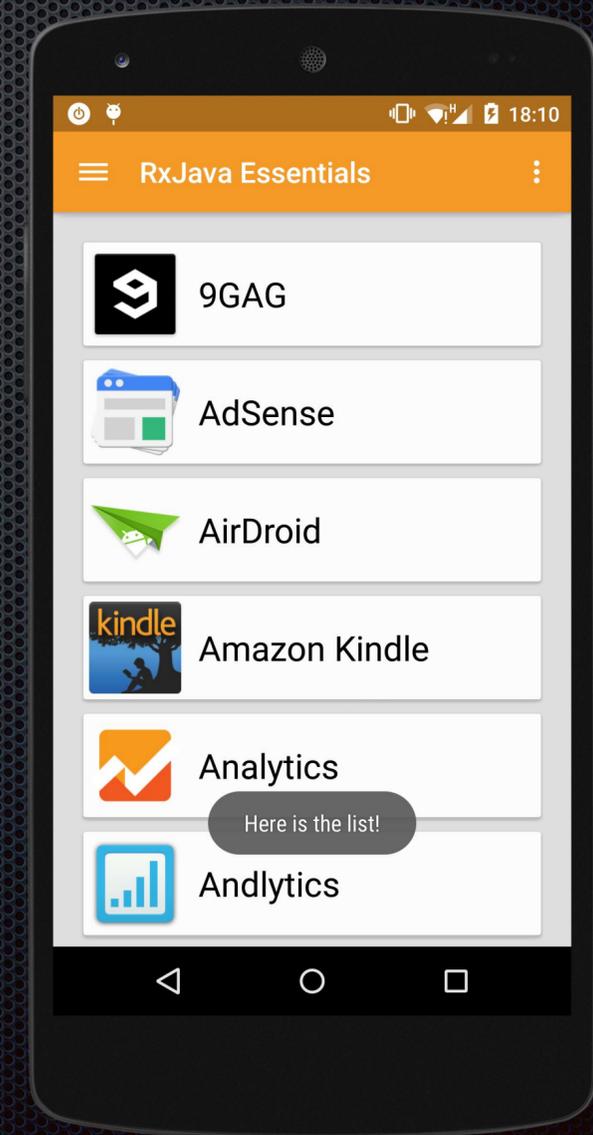


Rx Observer Pattern



Please, show some Android

- **Standard Android app**
- **A bit of Material Design**
- **A RecyclerView**
- **A list of installed apps**



Create an Observable

```
private Observable<AppInfo> getApps() {
    return Observable.create(subscriber -> {
        List<AppInfoRich> apps = getList();

        for (AppInfoRich appInfo : apps) {
            if (subscriber.isUnsubscribed()) {
                return;
            }
            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());
            String name = appInfo.getName();
            String iconPath = mFilesDir + "/" + name;
            Utils.storeBitmap(App.instance, icon, name);
            subscriber.onNext(
                new AppInfo(name, iconPath, appInfo.getLastUpdateTime()));
        }
        if (!subscriber.isUnsubscribed()) {
            subscriber.onCompleted();
        }
    });
}
```

Create an Observable

```
private Observable<AppInfo> getApps() {  
    return Observable.create(subscriber -> {  
        List<AppInfoRich> apps = getList();  
  
        for (AppInfoRich appInfo : apps) {  
            if (subscriber.isUnsubscribed()) {  
                return;  
            }  
            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());  
            String name = appInfo.getName();  
            String iconPath = mFilesDir + "/" + name;  
            Utils.storeBitmap(App.instance, icon, name);  
            subscriber.onNext(  
                new AppInfo(name, iconPath, appInfo.getLastUpdateTime()));  
        }  
        if (!subscriber.isUnsubscribed()) {  
            subscriber.onCompleted();  
        }  
    });  
}
```

Create an Observable

```
private Observable<AppInfo> getApps() {  
    return Observable.create(subscriber -> {  
        List<AppInfoRich> apps = getList();  
  
        for (AppInfoRich appInfo : apps) {  
            if (subscriber.isUnsubscribed()) {  
                return;  
            }  
            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());  
            String name = appInfo.getName();  
            String iconPath = mFilesDir + "/" + name;  
            Utils.storeBitmap(App.instance, icon, name);  
            AppInfo app = new AppInfo(name, iconPath, appInfo.getLastUpdateTime())  
            subscriber.onNext(app);  
        }  
        if (!subscriber.isUnsubscribed()) {  
            subscriber.onCompleted();  
        }  
    });  
}
```

Create an Observable

```
private Observable<AppInfo> getApps() {  
    return Observable.create(subscriber -> {  
        List<AppInfoRich> apps = getList();  
  
        for (AppInfoRich appInfo : apps) {  
            if (subscriber.isUnsubscribed()) {  
                return;  
            }  
            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());  
            String name = appInfo.getName();  
            String iconPath = mFilesDir + "/" + name;  
            Utils.storeBitmap(App.instance, icon, name);  
            AppInfo app = new AppInfo(name, iconPath, appInfo.getLastUpdateTime())  
            subscriber.onNext(app);  
        }  
        if (!subscriber.isUnsubscribed()) {  
            subscriber.onCompleted();  
        }  
    });  
}
```

Create an Observable

```
private Observable<AppInfo> getApps() {  
    return Observable.create(subscriber -> {  
        List<AppInfoRich> apps = getList();  
  
        for (AppInfoRich appInfo : apps) {  
            if (subscriber.isUnsubscribed()) {  
                return;  
            }  
            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());  
            String name = appInfo.getName();  
            String iconPath = mFilesDir + "/" + name;  
            Utils.storeBitmap(App.instance, icon, name);  
            AppInfo app = new AppInfo(name, iconPath, appInfo.getLastUpdateTime())  
            subscriber.onNext(app);  
        }  
        if (!subscriber.isUnsubscribed()) {  
            subscriber.onCompleted();  
        }  
    });  
}
```

`.create()` an Observable



With great power comes great responsibility.

I already have a list

I already have a list

Observable.from()

```
List<AppInfo> appsList = [...]  
Observable.from(appsList)  
    .subscribe(...)
```

What about just three elements?

What about just three elements?

Observable.just()

```
AppInfo appOne = appsList.get(0);  
AppInfo appTwo = appsList.get(1);  
AppInfo appThree = appsList.get(2);
```

```
Observable.just(appOne, appTwo, appThree)  
    .subscribe(...)
```

From old code to Rx

```
public boolean doSomething() {  
    // do something, then  
    return true;  
}
```

```
public Observable<Boolean> rxDoSomething() {  
    return Observable.fromCallable(this::doSomething);  
}
```

```
rxDoSomething().subscribe(...);
```

Subscribe and react

```
recyclerView.setLayoutManager(new LinearLayoutManager(view.getContext()));
```

```
adapter = new ApplicationAdapter(new ArrayList<>(), R.layout.applications_list_item);  
recyclerView.setAdapter(adapter);
```

```
SwipeRefreshLayout.setColorSchemeColors(mResources.getColor(R.color.myPrimaryColor));  
SwipeRefreshLayout.setProgressViewOffset(false, 0,  
    (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 24, [...]));
```

```
SwipeRefreshLayout.setEnabled(false);  
SwipeRefreshLayout.setRefreshing(true);
```

```
Observable.from(appsList).subscribe(observer);
```

Subscribe and react

```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onNext(AppInfo appInfo) {  
        addedApps.add(appInfo);  
        adapter.addApplication(addedApps.size() - 1, appInfo);  
    }  
  
    @Override  
    public void onCompleted() {  
        showCompleteSequenceToast();  
        stopProgressBar();  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        showErrorToast();  
        stopProgressBar();  
    }  
};
```

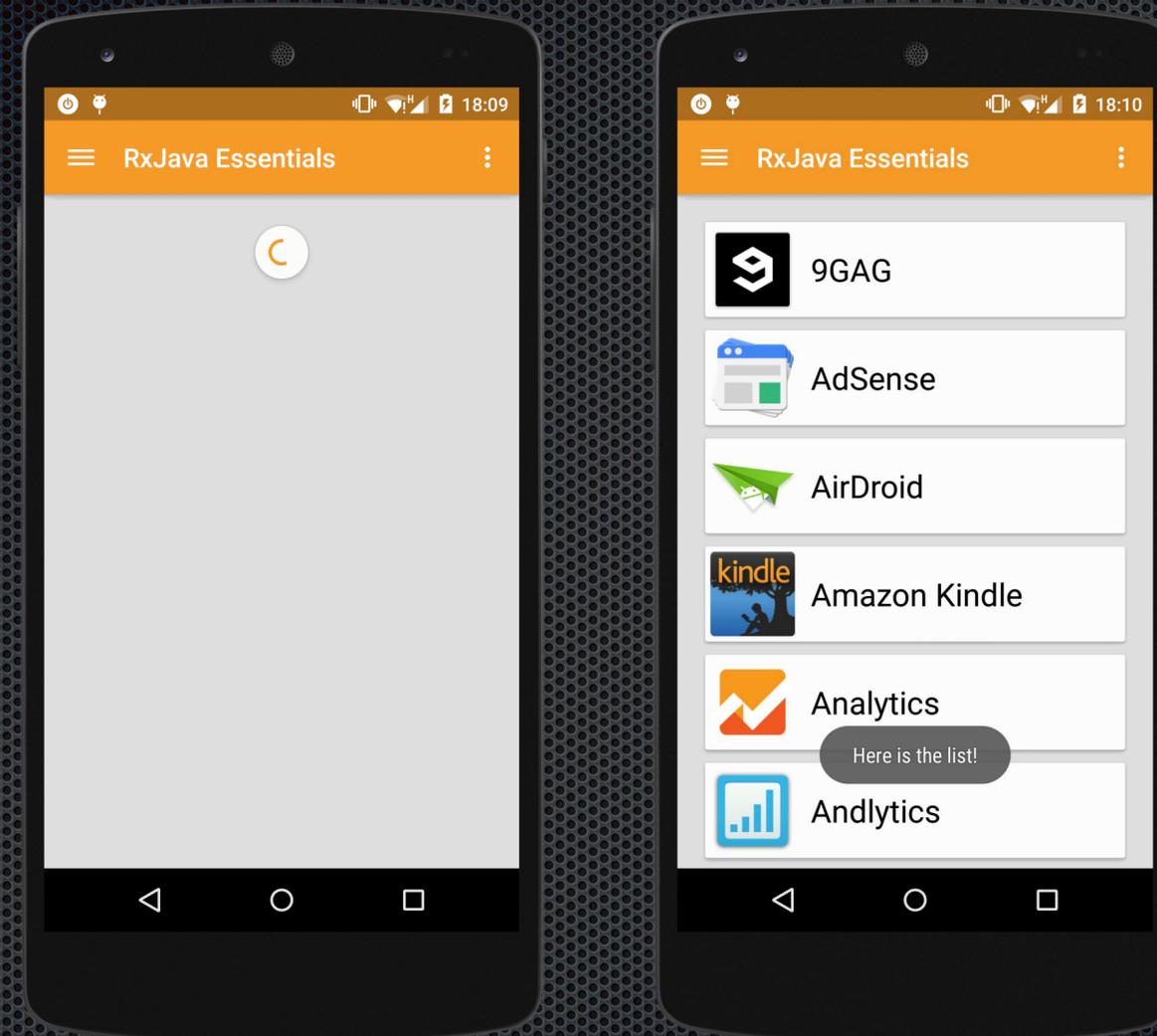
Subscribe and react

```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onNext(AppInfo appInfo) {  
        addedApps.add(appInfo);  
        adapter.addApplication(addedApps.size() - 1, appInfo);  
    }  
  
    @Override  
    public void onCompleted() {  
        showCompleteSequenceToast();  
        stopProgressBar();  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        showErrorToast();  
        stopProgressBar();  
    }  
};
```

Subscribe and react

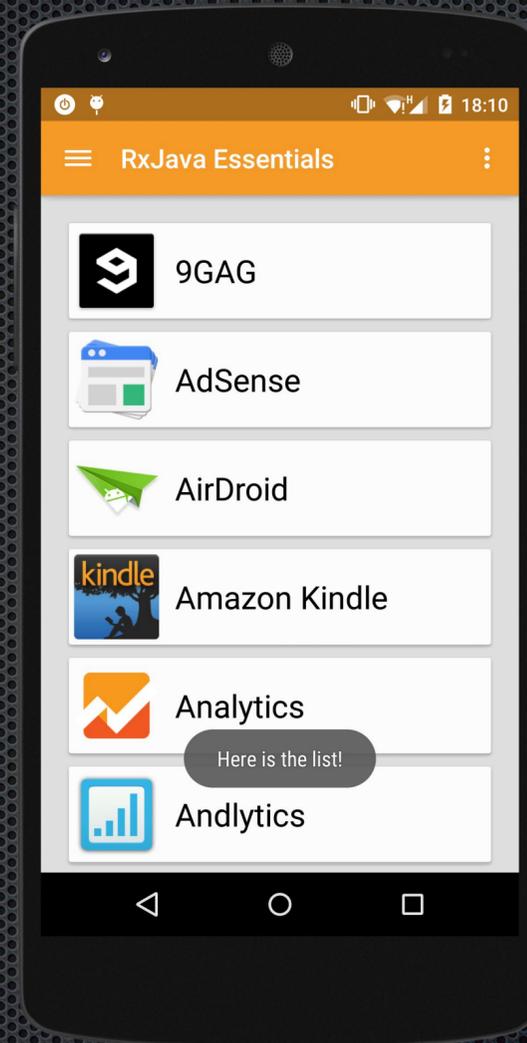
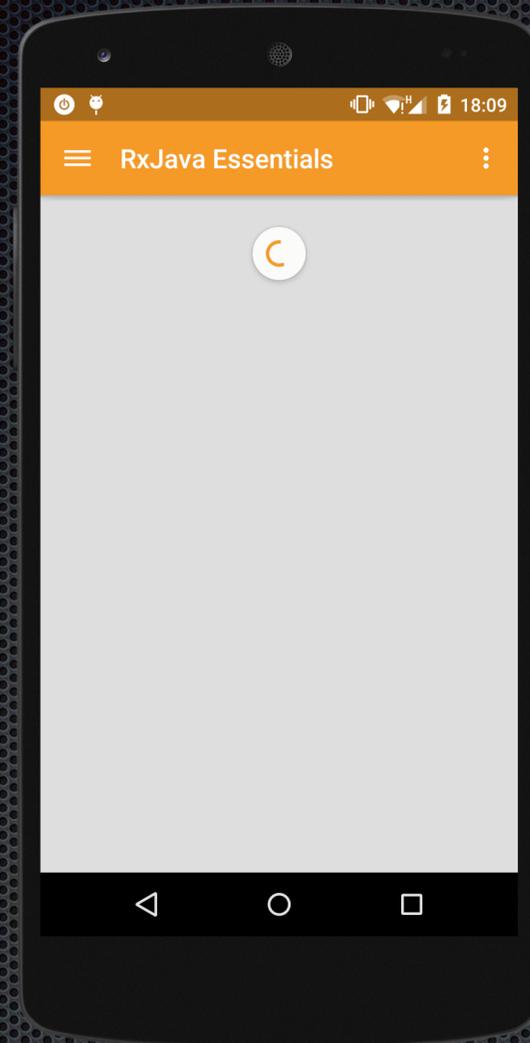
```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onNext(AppInfo appInfo) {  
        addedApps.add(appInfo);  
        adapter.addApplication(addedApps.size() - 1, appInfo);  
    }  
  
    @Override  
    public void onCompleted() {  
        showCompleteSequenceToast();  
        stopProgressBar();  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        showErrorToast();  
        stopProgressBar();  
    }  
};
```

Subscribe and react



Subscribe and react

I'm not
buying it!!



Show
more!!!!

Filtering

```
incomingApps()  
.subscribe(observer);
```

```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onCompleted() {  
        [...]  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        [...]  
    }  
  
    @Override  
    public void onNext(AppInfo appInfo) {  
        [...]  
    }  
};
```

Filtering

```
incomingApps()  
.filter((appInfo) -> appInfo.getName().startsWith("C"))  
.subscribe(observer);
```

```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onCompleted() {  
        [...]  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        [...]  
    }  
  
    @Override  
    public void onNext(AppInfo appInfo) {  
        [...]  
    }  
};
```

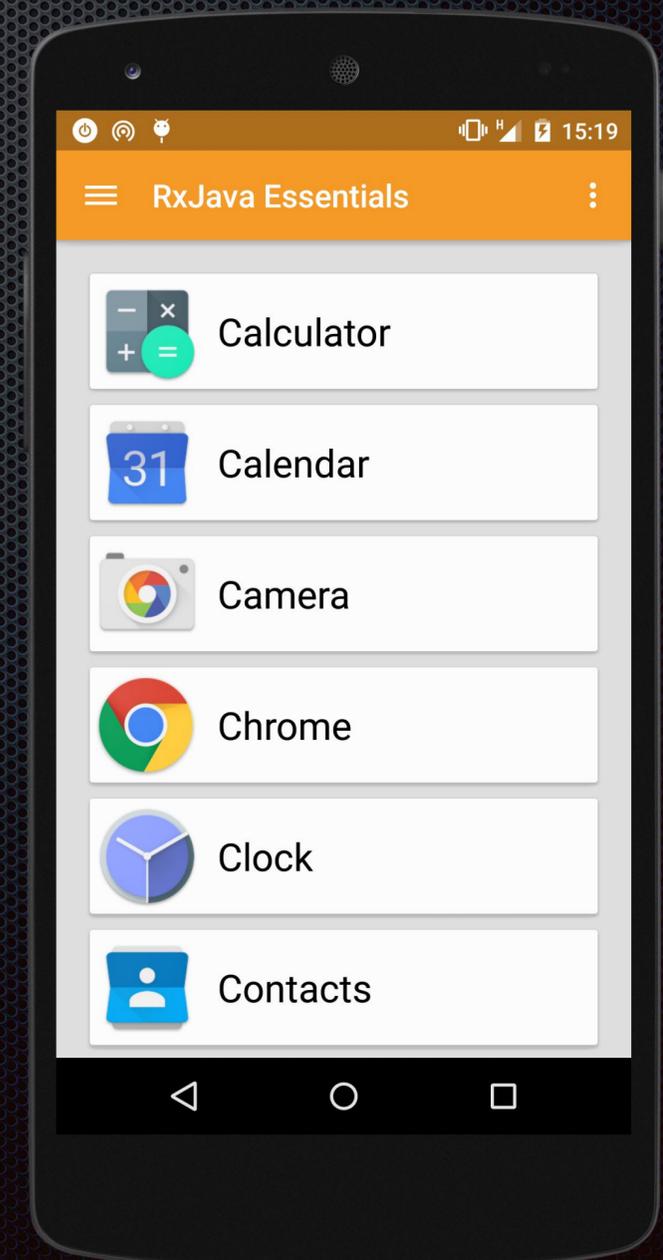
Filtering

```
incomingApps()
```

```
.filter((appInfo) -> appInfo.getName().startsWith("C"))
```

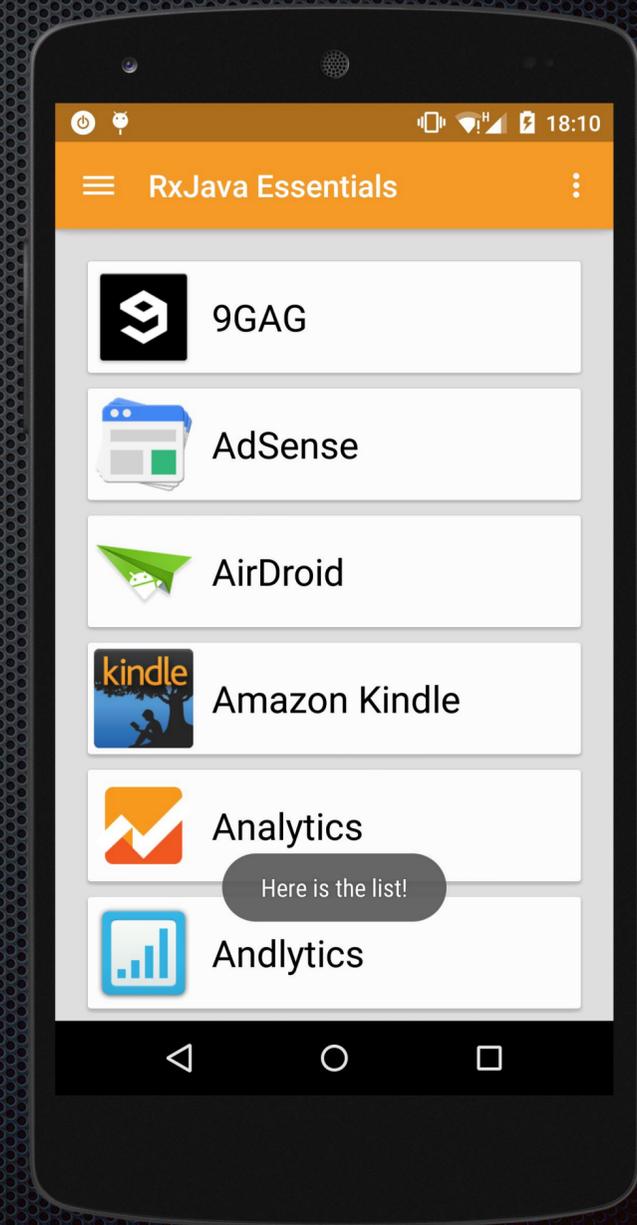
```
.subscribe(observer);
```

```
Observer<AppInfo> observer = new Observer<AppInfo>() {  
    @Override  
    public void onCompleted() {  
        [...]  
    }  
  
    @Override  
    public void onError(Throwable e) {  
        [...]  
    }  
  
    @Override  
    public void onNext(AppInfo appInfo) {  
        [...]  
    }  
};
```



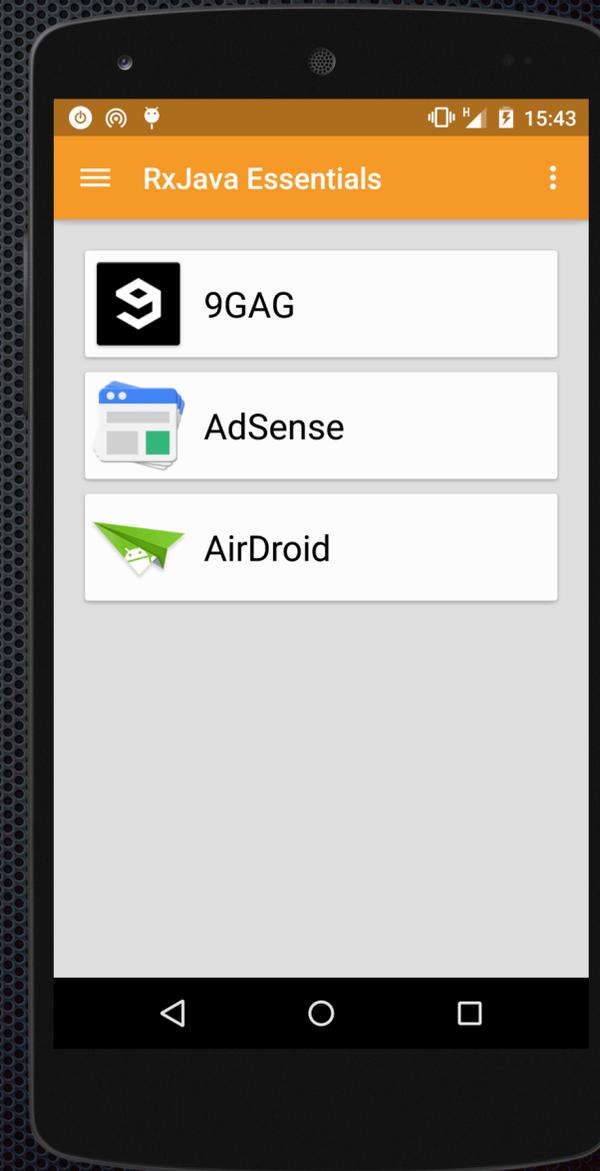
tl;dr;

```
incomingApps()  
.subscribe(observer);
```



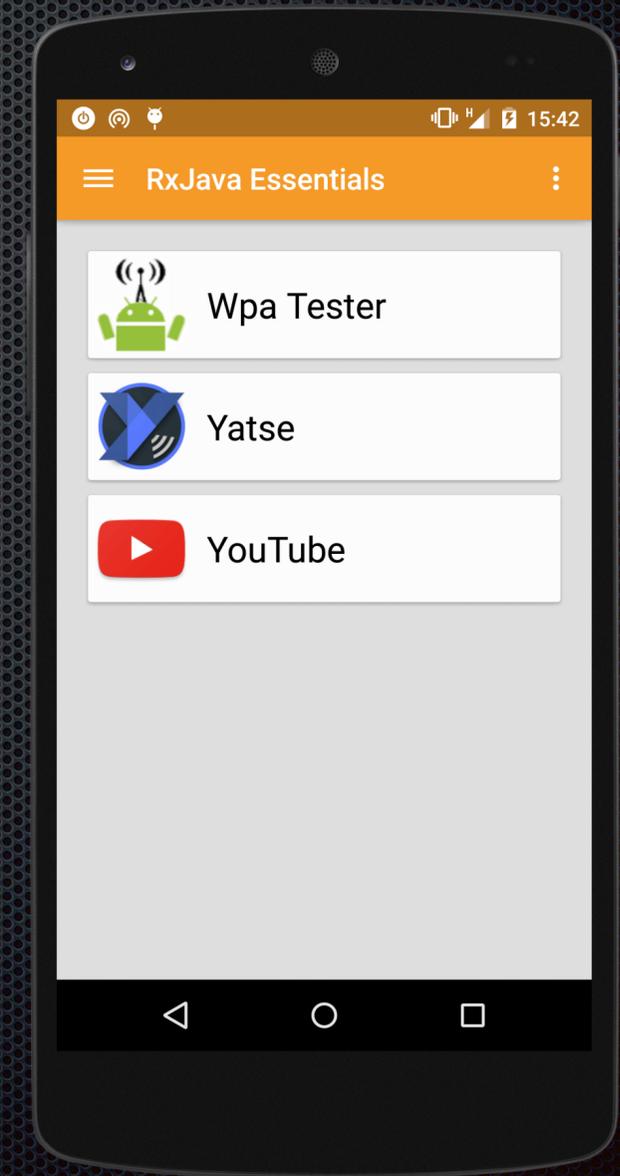
tl;dr;

```
incomingApps()  
.take(3)  
.subscribe(observer);
```



This is The End

```
incomingApps()  
  .takeLast(3)  
  .subscribe(jonSnow);
```



I hate duplicates!

```
incomingApps()
```

```
  .take(3)
```

```
  .repeat(3)
```

```
  .distinct()
```

```
  .subscribe(jonSnow);
```

-25... -25... -25... -25...

```
currentTemperature()  
.distinctUntilChanged()  
.subscribe(jonSnowAtTheWall);
```

-25..... -26.....

```
currentTemperature()  
  .sample(5, TimeUnit.MINUTES)  
    .distinctUntilChanged()  
    .subscribe(jonSnowAtTheWall);
```

-25..... -26.....

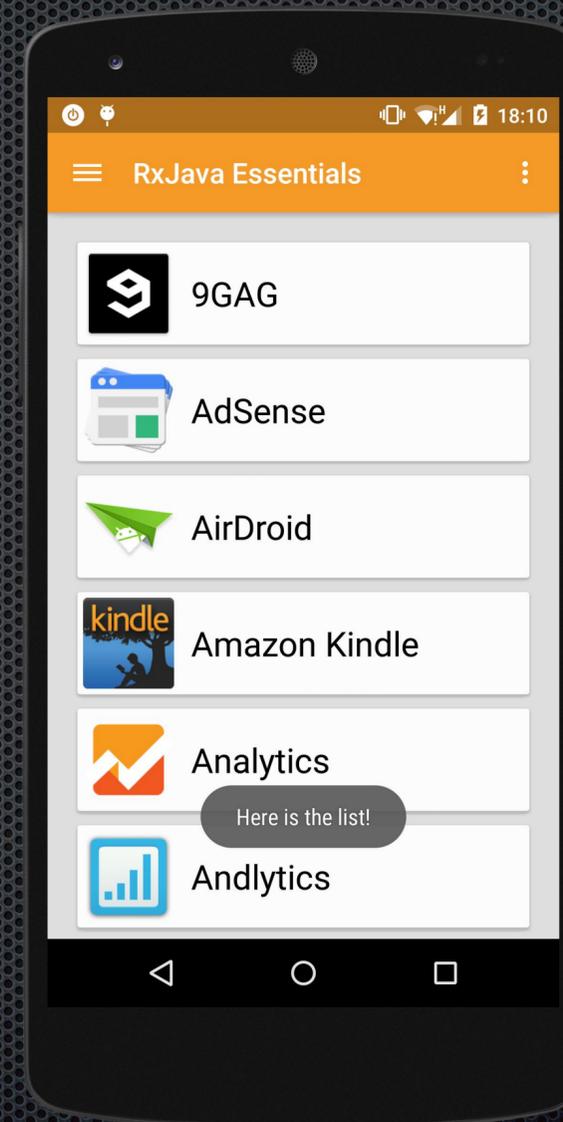
currentTemperature()

.throttleFirst(5, TimeUnit.MINUTES)

.subscribe(jonSnowAtTheWall);

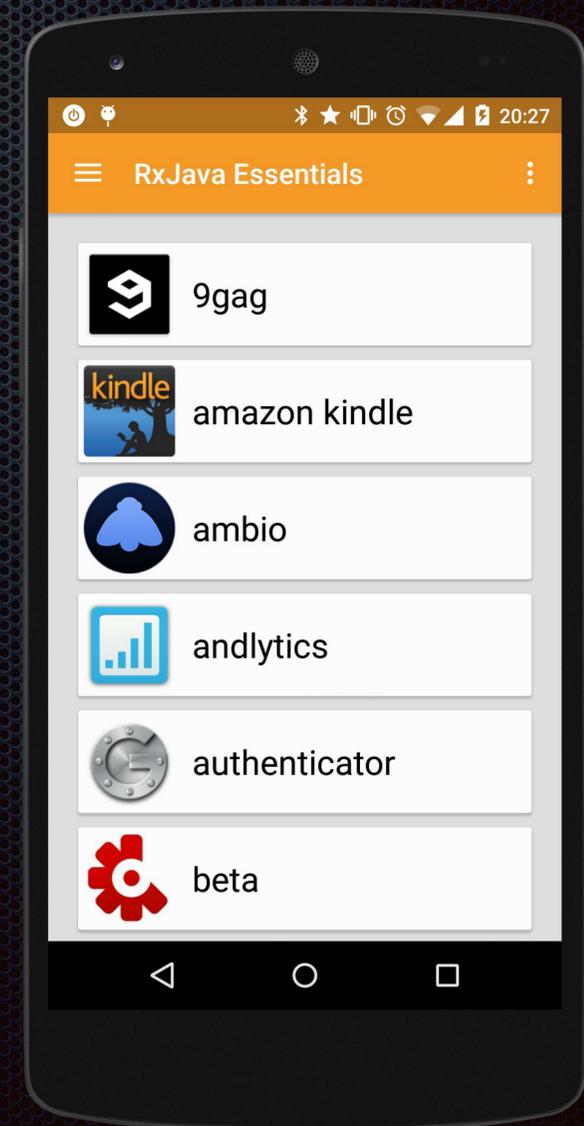
Transforming

```
incomingApps()  
.subscribe(jonSnowIsBack);
```



Transforming

```
incomingApps()  
  .map((AppInfo appInfo) -> {  
    appInfo.setName(  
      appInfo.getName().toLowerCase());  
    return appInfo;  
  })  
  .subscribe(jonSnowIsBack);
```



Transforming

- 📌 FlatMap
- 📌 SwitchMap
- 📌 ConcatMap
- 📌 Cast `.cast(Long.class)`
- 📌 Buffer `.buffer(3)`

Combining

```
List rev = Lists.reverse(apps);  
Observable<AppInfo> apps = Observable.from(apps);  
Observable<AppInfo> rApps = Observable.from(rev);
```

Combining

```
List rev = Lists.reverse(apps);  
Observable<AppInfo> apps = Observable.from(apps);  
Observable<AppInfo> rApps = Observable.from(rev);
```

Observable

```
.merge(apps, rApps)  
.subscribe(jonSnow);
```

Combining

```
Observable<AppInfo> apps = Observable.from(list);  
Observable<Long> tictoc = Observable.interval(1, TimeUnit.SECONDS);  
  
private AppInfo updateTitle(AppInfo appInfo, Long time) {  
    appInfo.setName(time + " " + appInfo.getName());  
    return appInfo;  
}
```

Combining

```
Observable<AppInfo> apps = Observable.from(list);  
Observable<Long> tictoc = Observable.interval(1, TimeUnit.SECONDS);  
  
private AppInfo updateTitle(AppInfo appInfo, Long time) {  
    appInfo.setName(time + " " + appInfo.getName());  
    return appInfo;  
}
```

Observable

```
.zip(apps, tictoc, updateTitle)  
.subscribe(jonSnow)
```

Defeating Android MainThread issue

```
private Observable<List<AppInfo>> getAppsList() {
    return Observable.create(subscriber -> {
        List<AppInfo> apps = new ArrayList<>();
        SharedPreferences sharedPref =
            getActivity().getPreferences(Context.MODE_PRIVATE);
        Type appInfoType = new TypeToken<List<AppInfo>>().getType();
        String serializedApps = sharedPref.getString("APPS", "");
        if (!"".equals(serializedApps)) {
            apps = new Gson().fromJson(serializedApps, appInfoType);
        }
        subscriber.onNext(apps);
        subscriber.onCompleted();
    });
}
```

Defeating Android MainThread issue

```
private Observable<List<AppInfo>> getAppsList() {
    return Observable.create(subscriber -> {
        List<AppInfo> apps = new ArrayList<>();
        SharedPreferences sharedPref =
            getActivity().getPreferences(Context.MODE_PRIVATE);
        Type appInfoType = new TypeToken<List<AppInfo>>().getType();
        String serializedApps = sharedPref.getString("APPS", "");
        if (!"".equals(serializedApps)) {
            apps = new Gson().fromJson(serializedApps, appInfoType);
        }
        subscriber.onNext(apps);
        subscriber.onCompleted();
    });
}
```

Defeating Android MainThread issue

```
private Observable<List<AppInfo>> getAppsList() {
    return Observable.create(subscriber -> {
        List<AppInfo> apps = new ArrayList<>();
        SharedPreferences sharedPref =
            getActivity().getPreferences(Context.MODE_PRIVATE);
        Type appInfoType = new TypeToken<List<AppInfo>>().getType();
        String serializedApps = sharedPref.getString("APPS", "");
        if (!"".equals(serializedApps)) {
            apps = new Gson().fromJson(serializedApps, appInfoType);
        }
        subscriber.onNext(apps);
        subscriber.onCompleted();
    });
}
```

Defeating Android MainThread issue

**I WANT TO PLAY A
GAME**



#ANDROID #LEGACYCODE LET'S ENABLE STRICTMODE

Defeating Android MainThread issue

```
D/StrictMode: StrictMode policy violation; ~duration=253 ms:  
    android.os.StrictMode$StrictModeDiskReadViolation:  
policy=31violation=2 at android.os.StrictMode  
$AndroidBlockGuardPolicy.onReadFromDisk(StrictMode.java:1135)
```

```
getAppList()
```

```
.subscribeOn(Schedulers.io())
```

```
.subscribe(jonSnow)
```

Defeating Android MainThread issue

```
D/StrictMode: StrictMode policy violation; ~duration=253 ms:  
    android.os.StrictMode$StrictModeDiskReadViolation:  
policy=31violation=2 at android.os.StrictMode  
$AndroidBlockGuardPolicy.onReadFromDisk(StrictMode.java:1135)
```

```
    getAppList()  
        .subscribeOn(Schedulers.io())  
        .subscribe(jonSnow)
```

*Only the original thread that created
a view hierarchy can touch its views*

Defeating Android MainThread issue

```
D/StrictMode: StrictMode policy violation; ~duration=253 ms:  
    android.os.StrictMode$StrictModeDiskReadViolation: policy=31violation=2 at  
    android.os.StrictMode$AndroidBlockGuardPolicy.onReadFromDisk(StrictMode.java:1135)
```

```
    getAppList()  
    .subscribeOn(Schedulers.io())  
    .subscribe(new Observer<List<AppInfo>>() {...})
```

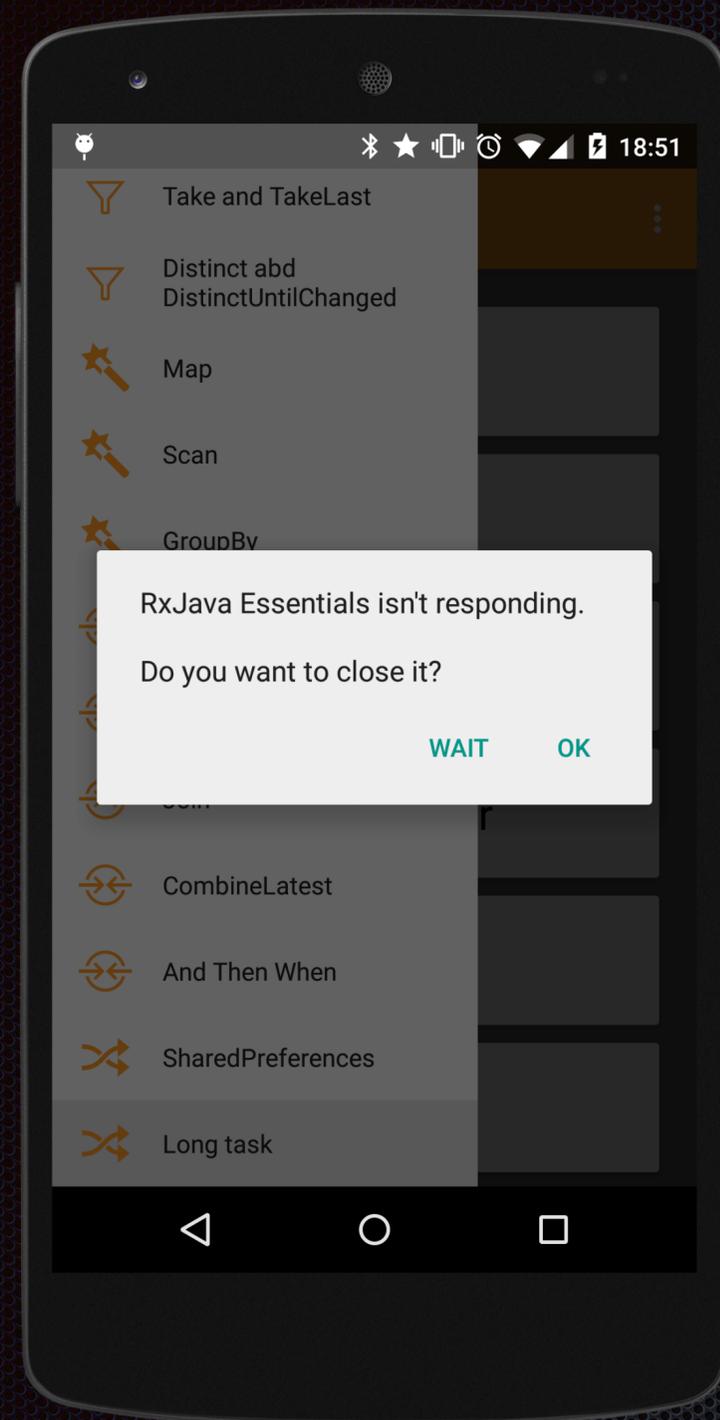
Only the original thread that created a view hierarchy can touch its views.

```
getAppList()  
    .subscribeOn(Schedulers.io())  
    .observeOn(AndroidSchedulers.mainThread())  
    .subscribe(jonSnow);
```

Defeating Android MainThread issue

```
private Observable<AppInfo> getApps(List<AppInfo> apps) {  
    return Observable.create(subscriber -> {  
        // let's waste some time  
        for (double i = 0; i < 10000000000; i++) {  
            double y = i * i;  
        }  
  
        for (AppInfo app : apps) {  
            subscriber.onNext(app);  
        }  
        subscriber.onCompleted();  
    });  
}
```

Defeating Android MainThread issue



I/Choreographer: Skipped 598 frames!
The application may be doing too much
work on its main thread.

Schedule all the things!!

```
getObservableApps ( apps )  
  .subscribeOn ( Schedulers.computation ( ) )  
  .observeOn ( AndroidSchedulers.mainThread ( ) )  
  .subscribe ( jonShow )
```

- ✓ Schedulers.immediate ()
- ✓ Schedulers.trampoline ()
- ✓ Schedulers.newThread ()

Conclusions

Observable sequences are like rivers: they **flow**.

You can **filter** a river, you can **transform** a river, you can **combine** two rivers into one and it will still flow.

In the end, it will be the river **you want** it to be.

Be water, my friend.

- Bruce Lee