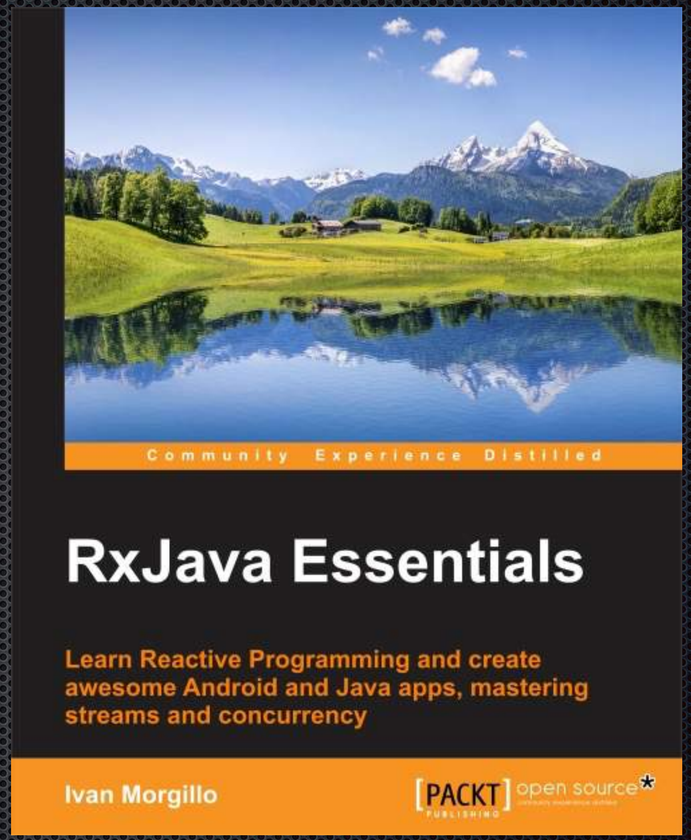
Ivan Morgillo

Author of RxJava Essentials by Packt Publishing



Author of Gratis Ebooks for Kindle

Android Reactive Programming with RxJava

Reactive Programming

- Erik Meijer - Rx .Net

- Ben Christensen - RxJava

RxJava

- Data flow
- Observer pattern
- Push vs Pull

Observer Pattern

- Notifications
- Automation
- Cause-Effect

RxJava Observer Pattern

- Observable
- Observer
- Subscriber
- Subject

RxJava Observer Pattern

- Observable
- Observer
- Subscriber
- Subject

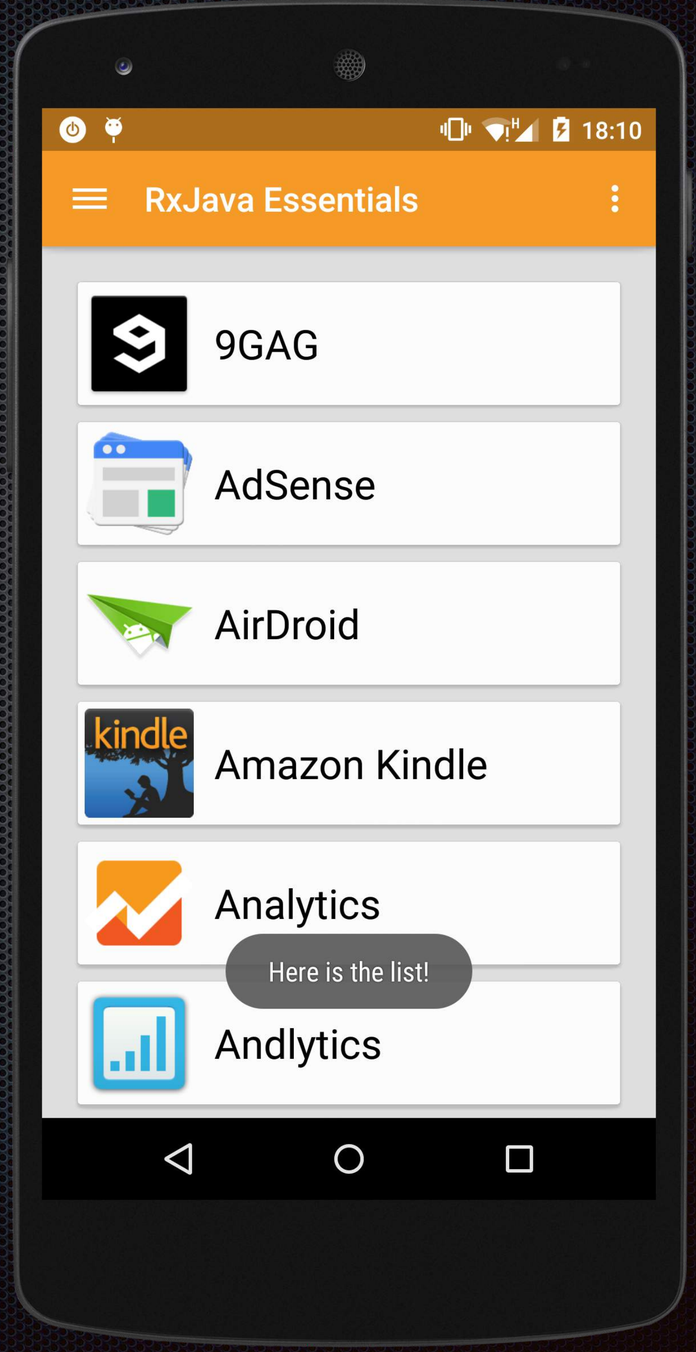
- OnNext()
- OnError()
- OnCompleted()

RxJava Observer Pattern

Observable		Observer
item1	onNext(item1)	Use item1
item2	onNext(item2)	Use item2
item3	onNext(item3)	Use item3

Please, show some Android

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



Create an Observable

create()

subscriber.onNext()

subscriber.onCompleted()

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

                    final Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
                    mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

                    List<ResolveInfo> infos = getActivity().getPackageManager()
                                                .queryIntentActivities(mainIntent, 0);
                    for (ResolveInfo info : infos) {
                        apps.add(new AppInfoRich(getActivity(), info));
                    }

                    for (AppInfoRich appInfo : apps) {
                        Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());
                        String name = appInfo.getName();
                        String iconPath = mFilesDir + "/" + name;
                        Utils.storeBitmap(App.instance, icon, name);

                        if (subscriber.isUnsubscribed()) {
                            return;
                        }
                        subscriber.onNext(new AppInfo(name, iconPath, appInfo.getLastUpdateTime()));
                    }
                    if (!subscriber.isUnsubscribed()) {
                        subscriber.onCompleted();
                    }
                });
    }
```

Create an Observable

create()

```
private Observable<AppInfo> getApps() {
        return Observable
                .create(new Observable.OnSubscribe<AppInfo>() {
                    @Override
                    public void call(Subscriber<? super AppInfo> subscriber) {
                        List<AppInfoRich> apps = new ArrayList<>();

                        final Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
                        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

                        List<ResolveInfo> infos = getActivity().getPackageManager().queryIntentActivities(mainIntent, 0);
                        for (ResolveInfo info : infos) {
                            apps.add(new AppInfoRich(getActivity(), info));
                        }

                        for (AppInfoRich appInfo : apps) {
                            Bitmap icon = Utils.drawableToBitmap(appInfo.getIcon());
                            String name = appInfo.getName();
                            String iconPath = mFilesDir + "/" + name;
                            Utils.storeBitmap(App.instance, icon, name);

                            if (subscriber.isUnsubscribed()) {
                                return;
                            }
                            subscriber.onNext(new AppInfo(name, iconPath, appInfo.getLastUpdateTime()));
                        }
                        if (!subscriber.isUnsubscribed()) {
                            subscriber.onCompleted();
                        }
                    }
                });
    }
```

subscriber.onNext()

subscriber.onCompleted()

Create an Observable

Observable.from()

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

Observable.just()

```
List<AppInfo> apps = ApplicationsList.getInstance().getList();

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

Observable<AppInfo> threeOfThem =
          Observable.just(appOne, appTwo, appThree);

threeOfThem.subscribe(...)
```

Observable.empty()

Observable.never()

Observable.throw()

Subscribe and react

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

mAdapter = new ApplicationAdapter(new ArrayList<>(), R.layout.applications_list_item);
mRecyclerView.setAdapter(mAdapter);

mSwipeRefreshLayout.setColorSchemeColors(getResources().getColor(R.color.myPrimaryColor));
mSwipeRefreshLayout.setProgressViewOffset(false, 0,
            (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 24,
getResources().getDisplayMetrics()));

// Progress
mSwipeRefreshLayout.setEnabled(false);
mSwipeRefreshLayout.setRefreshing(true);
mRecyclerView.setVisibility(View.GONE);
```

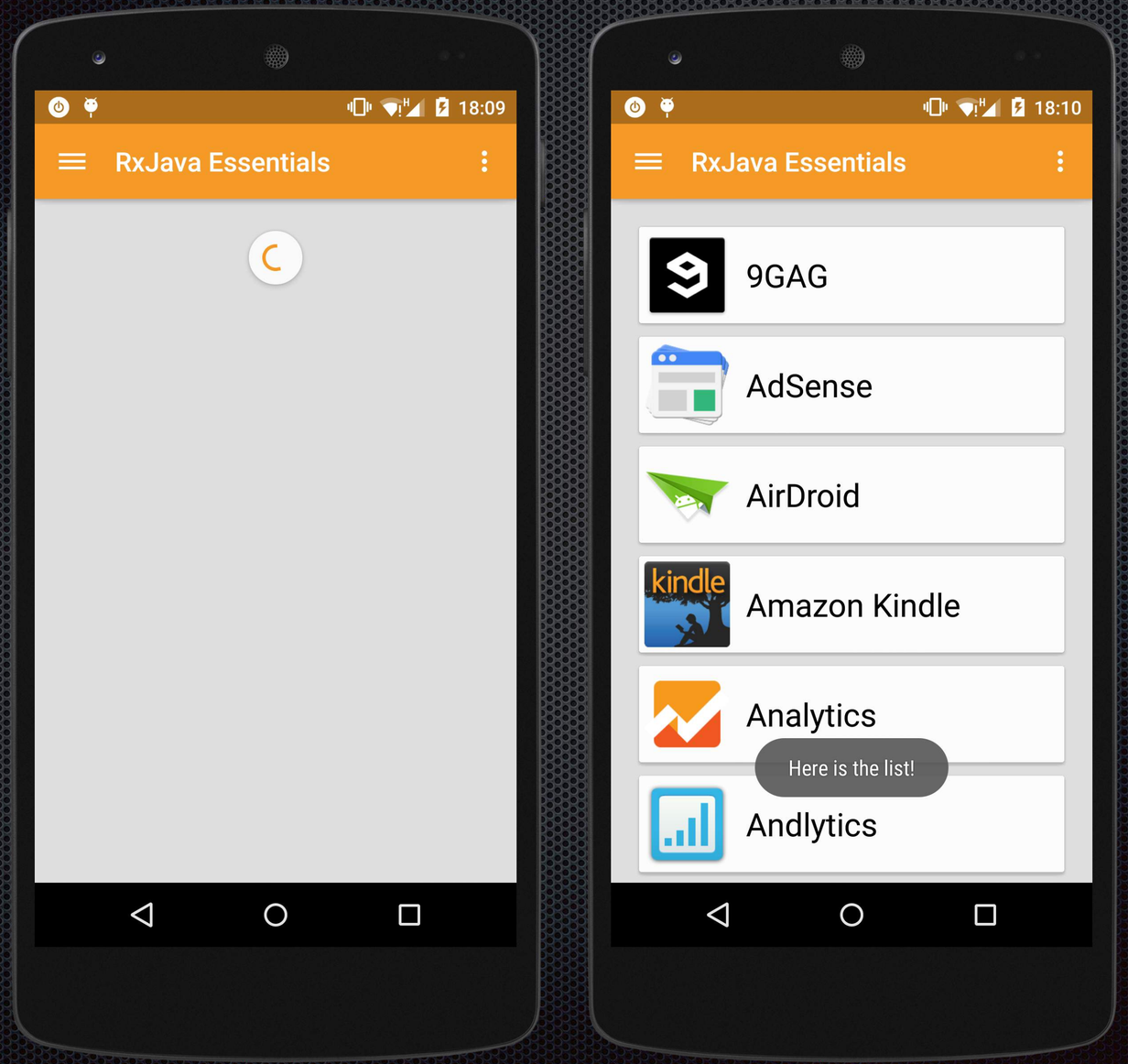
```
Observable.from(apps).subscribe(observer);
```

Subscribe and react

```
Observer<AppInfo> observer = new Observer<AppInfo>() {
        @Override
        public void onCompleted() {
            mSwipeRefreshLayout.setRefreshing(false);
            Toast.makeText(getActivity(), "Here is the list!", Toast.LENGTH_LONG).show();
        }

        @Override
        public void onError(Throwable e) {
            Toast.makeText(getActivity(), "Something went wrong!", Toast.LENGTH_SHORT).show();
            mSwipeRefreshLayout.setRefreshing(false);
        }

        @Override
        public void onNext(AppInfo appInfo) {
            mAddedApps.add(appInfo);
            mAdapter.addApplication(mAddedApps.size() - 1, appInfo);
        }
    };
```

Subscribe and react

I'm not buying it!!



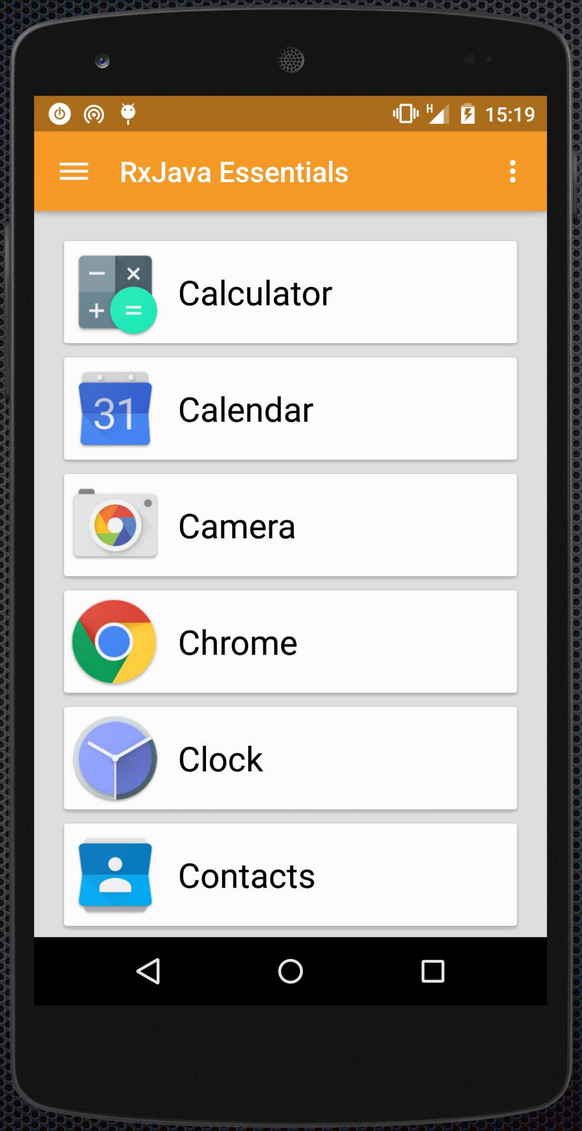
Show more!!!!

Filtering

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

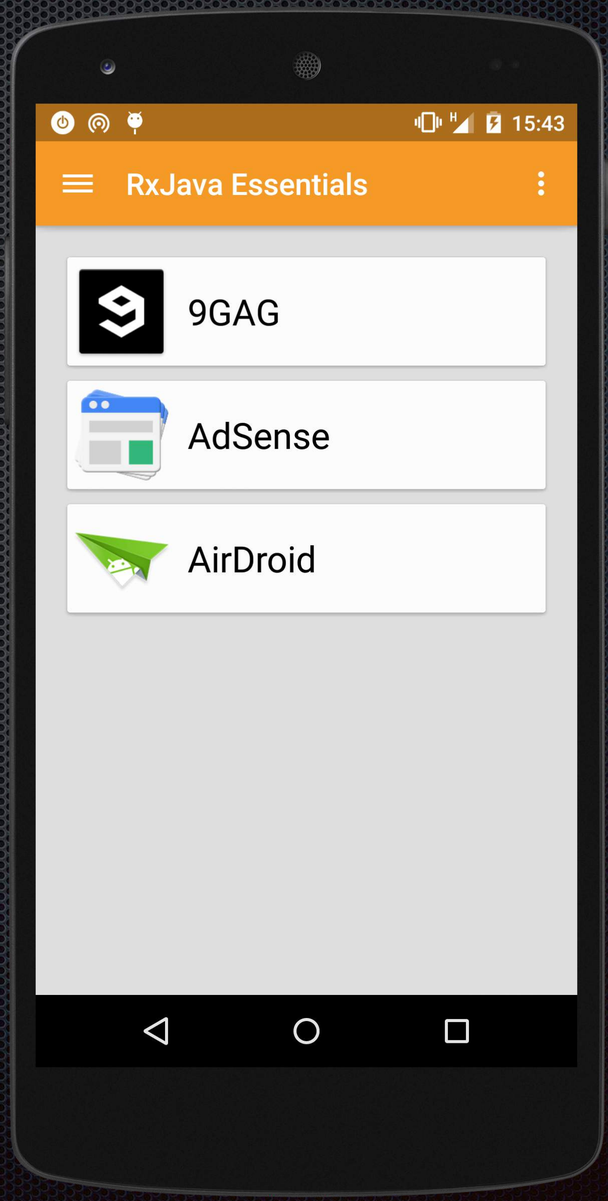
```
Observer<AppInfo> observer = new Observer<AppInfo>() {
            @Override
            public void onCompleted() {
                [...]
            }

            @Override
            public void onError(Throwable e) {
                [...]
            }

            @Override
            public void onNext(AppInfo appInfo) {
                [...]
            }
        };
```

Filtering


15:19
RxJava Essentials
Calculator
Calendar
Camera
Chrome
Clock
Contacts

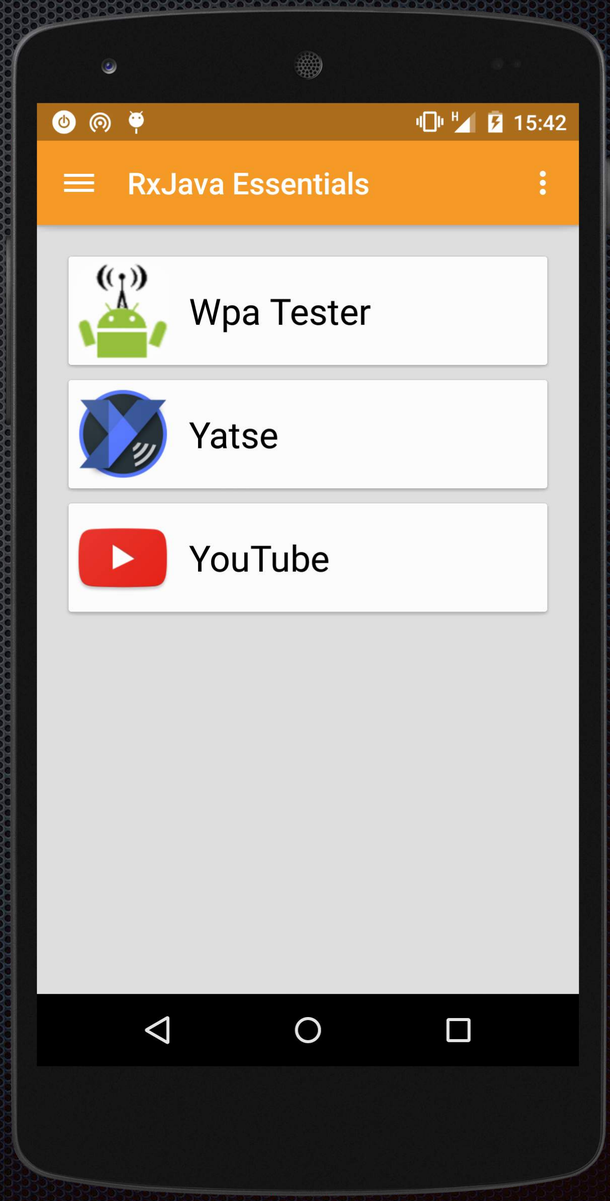
Take this!

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



Take this!

```
getApps()
.subscribeOn(io())
.takeLast(3)
.observeOn(mainThread())
.subscribe(observer);
```



I hate duplicates!

```
getApps()
  .take(3)
  .repeat(3)
  .distinct()
.subscribe(observer);
```

I hate duplicates!

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

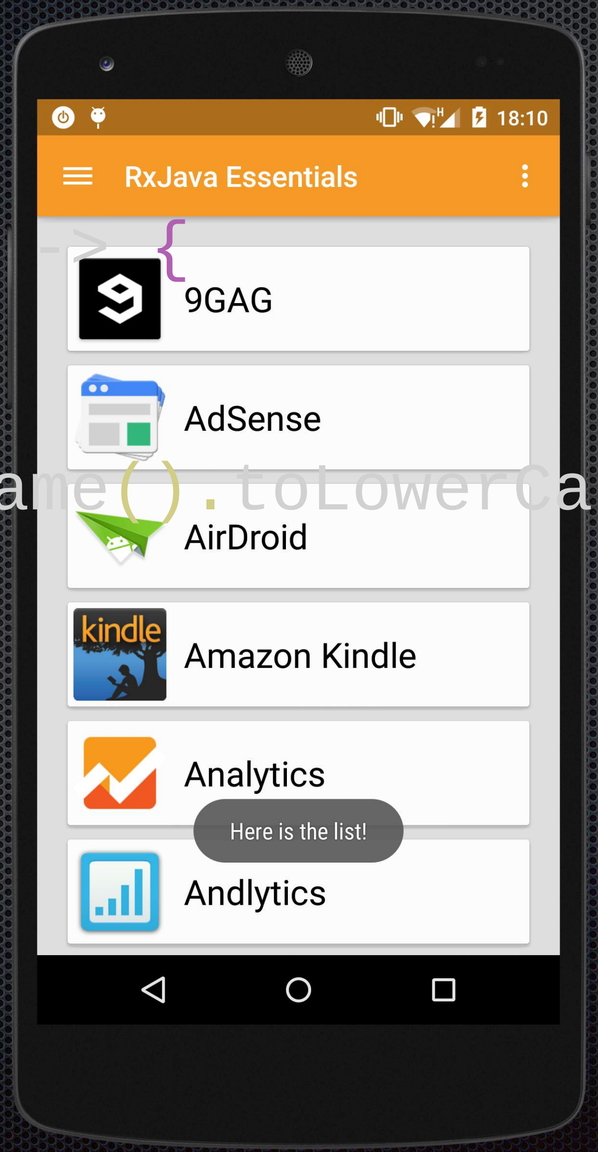
Loosen it up a bit

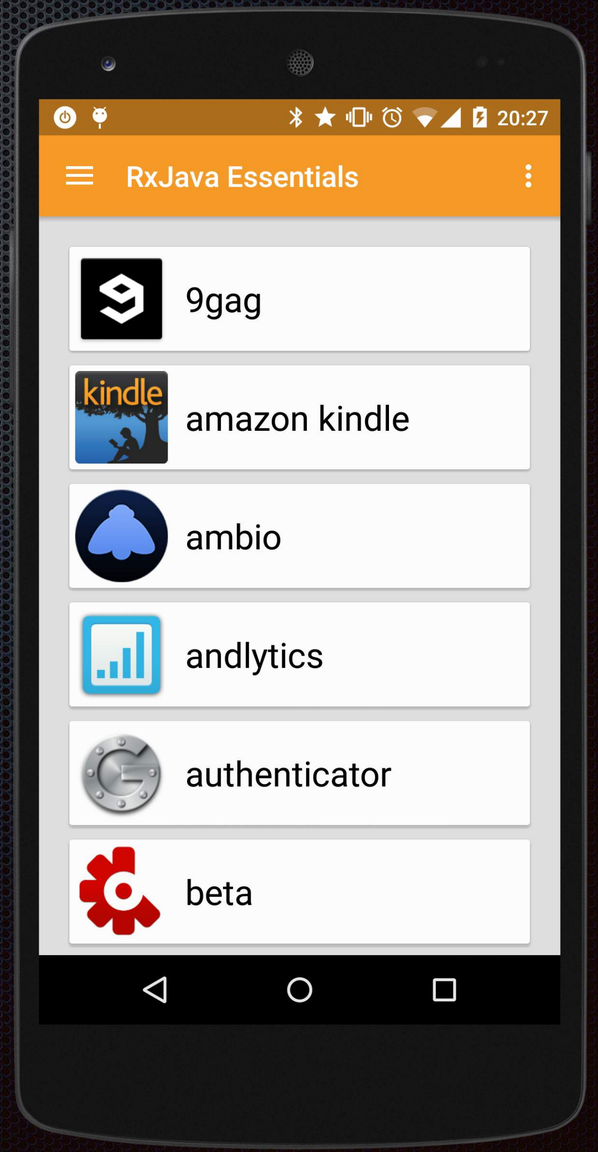
```
currentTemperature()
    .sample(30, TimeUnit.SECONDS)
    .subscribe(observer);
```

```
currentTemperature()
.throttleFirst(30, TimeUnit.SECONDS)
    .subscribe(observer)
```

Transforming

```
Observable
.from(apps)
.map(appInfo -> {
        appInfo.setName(
            appInfo.getName().toLowerCase());

        return appInfo;
})
```





Transformers

- FlatMap
- 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);
```

```
Observable
 .merge(apps, rApps)
 .subscribe(observer);
```

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, this::updateTitle)
.subscribe(observer)
```

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

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

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

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)
```

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

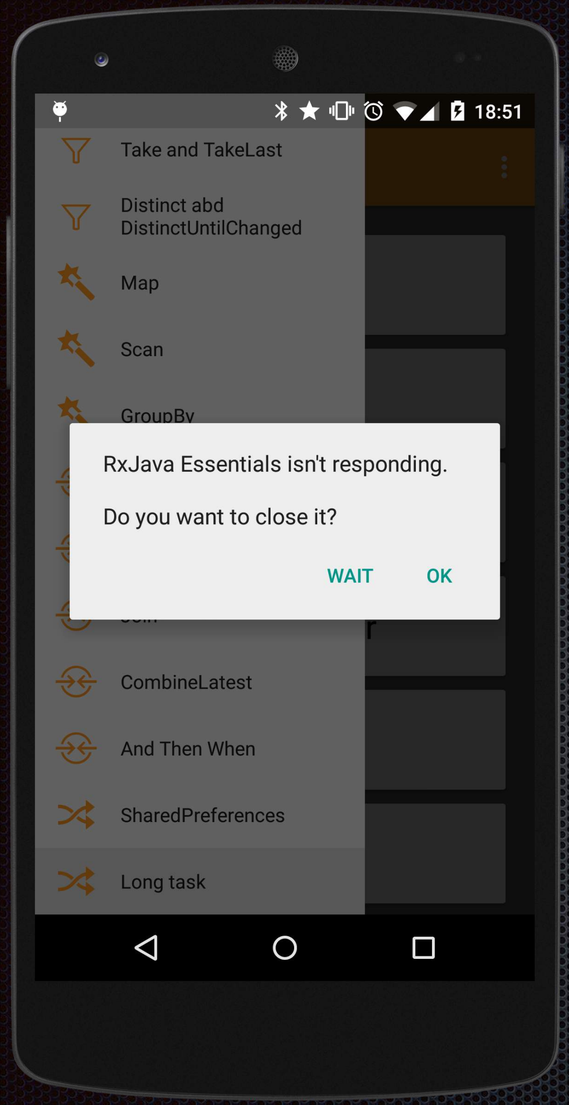
```
Only the original thread that created a view hierarchy can touch its views.
```

```
getAppsList()
.subscribeOn(Schedulers.io())
.observeOn(AndroidSchedulers.mainThread())
.subscribe(new Observer<List<AppInfo>>() {...}
```

Defeating Android MainThread issue

```
private Observable<AppInfo> getObservableApps(List<AppInfo> apps) {
        return Observable
                .create(subscriber -> {
                    for (double i = 0; i < 1000000000; 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.
```

Defeating Android MainThread issue

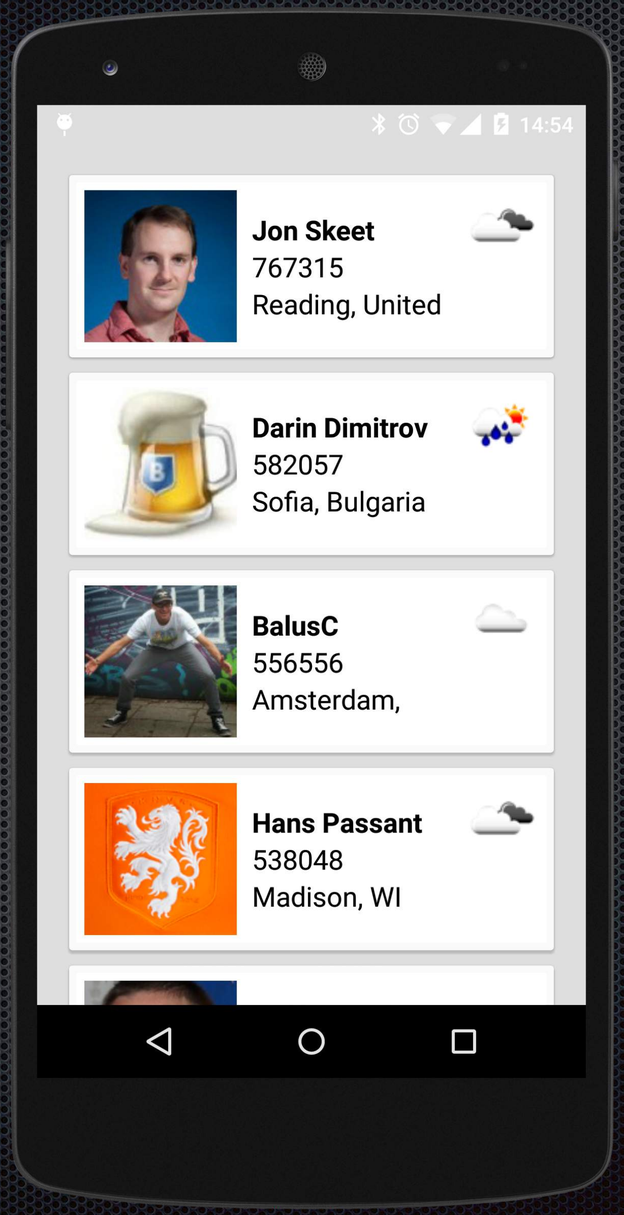
```
getObservableApps(apps)
.onBackpressureBuffer()
.subscribeOn(Schedulers.computation())
.observeOn(AndroidSchedulers.mainThread())
.subscribe(observer)
```

Welcome to the real world

Tools

- Retrolambda
- Butter Knife
- Lombok
- Retrofit
- Universal Image Loader

Welcome to the real world



Welcome to the real world

```
public interface StackExchangeService {

    @GET("/2.2/users?order=desc&sort=reputation&site=stackoverflow")
    Observable<UsersResponse> getMostPopularSOusers(@Query("pagesize") int howmany);
}


public interface OpenWeatherMapService {

    @GET("/data/2.5/weather")
    Observable<WeatherResponse> getForecastByCity(@Query("q") String city);
}
```

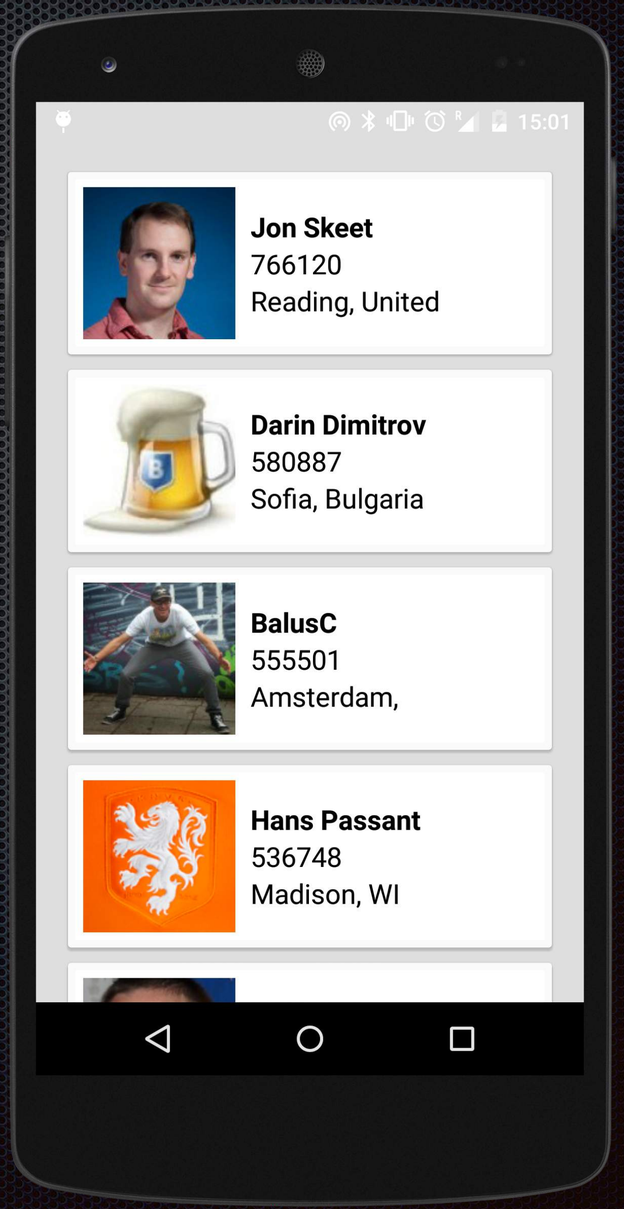
http://www.jsonschema2pojo.org/

Welcome to the real world

```
public class SeApiManager {

    private final StackExchangeService mStackExchangeService;

    public SeApiManager() {
        RestAdapter restAdapter = new RestAdapter.Builder()
                .setEndpoint("https://api.stackexchange.com")
                .setLogLevel(RestAdapter.LogLevel.BASIC)
                .build();

        mStackExchangeService = restAdapter.create(StackExchangeService.class);
    }

    public Observable<List<User>> getMostPopularSOusers(int howmany) {
        return mStackExchangeService
                .getMostPopularSOusers(howmany)
                .map(UsersResponse::getUsers)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
    }
}
```

Welcome to the real world

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    [...]
    mAdapter = new SoAdapter(new ArrayList<>());
    mAdapter.setOpenProfileListener(this);

    mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
    mRecyclerView.setAdapter(mAdapter);

    mSeApiManager = new SeApiManager();

    mSwipe.setOnRefreshListener(this::refreshList);

    refreshList();
}

private void refreshList() {
    showRefresh(true);
    mSeApiManager.getMostPopularSOusers(10)
            .subscribe(users -> {
                showRefresh(false);
                mAdapter.updateUsers(users);
            }, error -> {
                App.L.error(error.toString());
                showRefresh(false);
            });
}
```

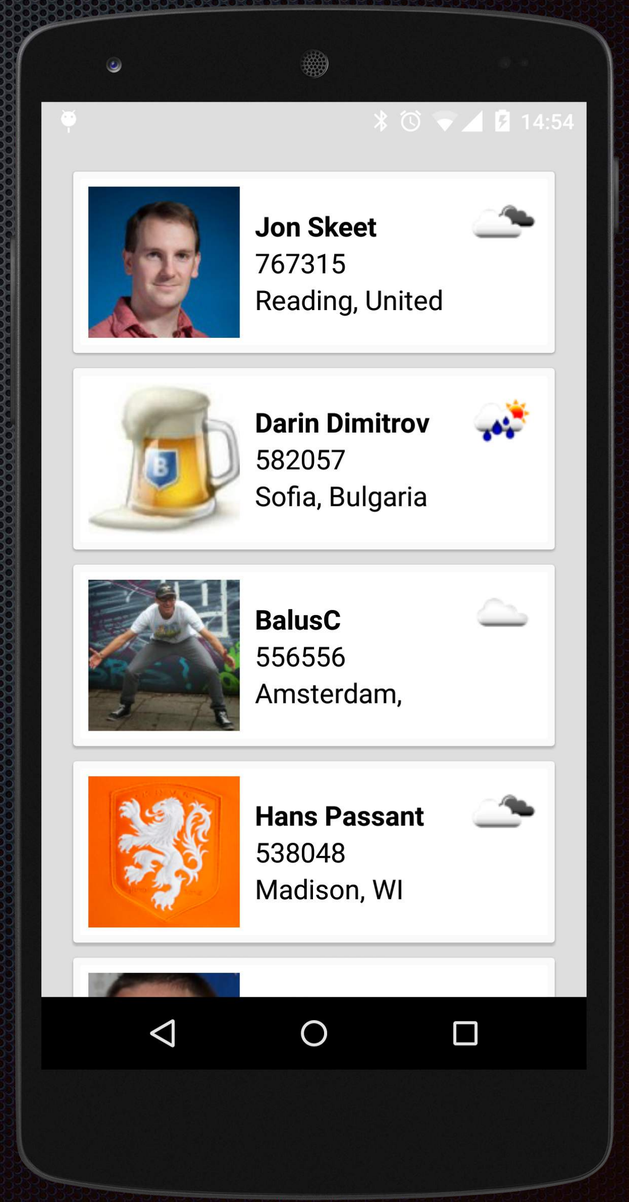


Welcome to the real world

```
public class OpenWeatherMapApiManager {

    @Getter
    private static OpenWeatherMapApiManager instance = new OpenWeatherMapApiManager();

    private final OpenWeatherMapService mOpenWeatherMapService;

    private OpenWeatherMapApiManager() {
        RestAdapter restAdapter = new RestAdapter.Builder()
                .setEndpoint("http://api.openweathermap.org")
                .setLogLevel(RestAdapter.LogLevel.BASIC)
                .build();

        mOpenWeatherMapService = restAdapter.create(OpenWeatherMapService.class);
    }

    public Observable<WeatherResponse> getForecastByCity(String city) {
        return mOpenWeatherMapService
                .getForecastByCity(city)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread());
    }
}
```

Welcome to the real world

```
private void displayWeatherInfos(User user) {
        [...]
      OpenWeatherMapApiManager.getInstance()
                              .getForecastByCity(city)
          .filter(response -> response != null)
          .filter(response -> response.getWeather().size() > 0)
          .concatMap(response -> {
              String url = getWeatherIconUrl(response);
              return loadBitmap(url);
          })
          .subscribeOn(Schedulers.io())
          .observeOn(AndroidSchedulers.mainThread())
          .subscribe(
              icon -> {
                 city_image.setImageBitmap(icon);
              },
              error -> {
                 App.L.error(error.toString());
              });
}}
```



Welcome to the real world

```
ViewObservable.clicks(mView)
    .subscribe(onClickEvent -> {
        checkNotNull(mProfileListener, "Must implement OpenProfileListener");

        String url = user.getWebsiteUrl();
        if (url != null && !url.equals("") && !url.contains("search")) {
            mProfileListener.open(url);
        } else {
            mProfileListener.open(user.getLink());
        }
});
```

Conclusions

http://reactivex.io

Observable sequences act 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