Ivan Morgillo

RxJava Essentials
by Packt Publishing

Gratis Ebooks for Kindle

GET IT ON Google play

Android Reactive Programming with RxJava

Reactive Programming

Erik Meijer - Rx .Net

Ben Christensen - RxJava

Reactive Programming

- Data flow
- Observer pattern
- Push vs Pull

Observer Pattern

- Notifications
- Automation
- Cause-Effect

Rx Observer Pattern

- Observable
- Observer
- Subscription
- Subject

Rx Observer Pattern

- Observable
- Observer
- Subscription
- Subject

- onNext()
- onError()
- onCompleted()

Rx Observer Pattern

Observable *emits*		Observer
item1	onNext(item1)	Uses item1
item2	onNext(item2)	Uses item2
item3	onNext(item3)	Uses item3

Time

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) {
            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

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

        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

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

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

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

Create an Observable

Observable.from()

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

Create an Observable

Observable.from()

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

Observable.just()

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

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

Create an Observable

Observable.from()

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

Observable.just()

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

Observable.empty()

Observable.never()

Observable.error()

Subscribe and react

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

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

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

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 onNext(AppInfo appInfo) {
            mAddedApps.add(appInfo);
            mAdapter.addApplication(mAddedApps.size() - 1, appInfo);
        }

        @Override
        public void onCompleted() {
            showCompleteSequenceToast();
        }

        @Override
        public void onError(Throwable e) {
            showErrorToast();
            stopProgressBar();
        }
    };
```

Subscribe and react

```
Observer<AppInfo> observer = new Observer<AppInfo>() {
        @Override
        public void onNext(AppInfo appInfo) {
            mAddedApps.add(appInfo);
            mAdapter.addApplication(mAddedApps.size() - 1, appInfo);
        }

        @Override
        public void onCompleted() {
            showCompleteSequenceToast();
        }

        @Override
        public void onError(Throwable e) {
            showErrorToast();
            stopProgressBar();
        }
    };
```

Subscribe and react

```
Observer<AppInfo> observer = new Observer<AppInfo>() {
        @Override
        public void onNext(AppInfo appInfo) {
            mAddedApps.add(appInfo);
            mAdapter.addApplication(mAddedApps.size() - 1, appInfo);
        }

        @Override
        public void onCompleted() {
            showCompleteSequenceToast();
        }

        @Override
        public void onError(Throwable e) {
            showErrorToast();
            stopProgressBar();
        }
    };
```

Subscribe and react

I'm not buying it!!

Show more!!!!

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);
.subscribe(jonSnow);
```

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(30, TimeUnit.SECONDS)
.subscribe(jonSnowAtTheWall);
```

Transforming

```
incomingApps()
.map(appInfo -> {
    appInfo.setName(
        appInfo.getName().toUpperCase());
    return appInfo;
})
```

Transforming

- 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(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;
}
```

```
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

```
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(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)
```

```
getAppsList()
.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)
```

```
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(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 < 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.
```

Schedule all the things!!

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

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

Welcome to the real world

Tools

- Retrolambda
- ButterKnife
- Lombok
- Retrofit
- Universal Image Loader

Welcome to the real world

REST in peace

```
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/

REST in peace

```
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());
    }
}
```

Android side

```
@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(); // Inject this with Dagger

    mSwipe.setOnRefreshListener(this::refreshList);

    refreshList();
}
```

Android side

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

Wrap and run

```
@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 -> {
                Log.e(error.getMessage());
                showRefresh(false);
            });
}
```

Add the forecast

```
private void displayWeatherInfos(User user) {
        [...]
            mOwmApiManager.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 -> {
                   cityImage.setImageBitmap(icon);
                },
                error -> {
                   Log.e(error.getMessage());
                });
}}
```

Add the forecast

```
private void displayWeatherInfos(User user) {
        [...]
            mOwmApiManager.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 -> {
                   cityImage.setImageBitmap(icon);
                },
                error -> {
                   Log.e(error.getMessage());
                });
}}
```

Add the forecast

```
private void displayWeatherInfos(User user) {
        [...]
            mOwmApiManager.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 -> {
                    cityImage.setImageBitmap(icon);
                },
                error -> {
                    Log.e(error.getMessage());
                });
}}
```

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

Contacts

```
Observable<String> username = Observable.just("hamen");

Observable<String> baseUrls =
    Observable.just("http://github.com/", "http://twitter.com/");

Observable
    .combineLatest(username, baseUrls, (user, social) -> social + user)
                .subscribe(url -> {
                    Log.i("CLICK_ME", url);
                });
```