Reactivex

with Kotlin

Jonathan Monga

Android Developer

Maishapay, Wenzeasy, CongoBD

JonathanMonga

@jonathan_monga

A little survey

A little survey

- Who are Android Developer ?
- Who are not Android Developer ?
- Who develops with Kotlin ?
- Who develops with Java ?
- Who heard about reactivex ?

Agenda

Agenda

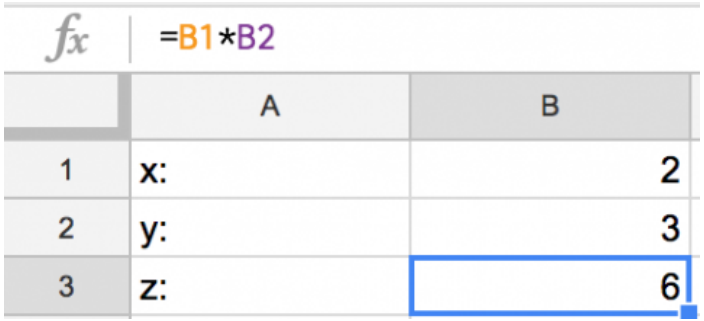
- Boring definitions
- Why RxKotlin ?
- Deep dive into RxKotlin

Boring definitions

Boring definitions

1) Reactive programming

« Is general term thas is focused on reacting to changes, such as data values or events. » Ben christensen.



	A	B
1	x:	10
2	y:	3
3	z:	30

Boring definitions

2) Reactives extensions

- An API to handle events synchronously or asynchronously through a flow of event.
- « Une bibliothèque permettant de composer des programmes asynchrones et basés sur des événements à l'aide de séquences observables. »

Boring definitions

3) What's RxJava or RxKotlin

- The java implementation of Reactive extensions.
- Une Une bibliothèque très légère qui apporte ou ajoute de fonctions d'extensions pratique à RxJava, vu l'interopérabilité entre Java et Kotlin, on peut utiliser RxJava directement avec Kotlin.

Boring definitions

4) Reactives extensions **(Once upon a time)**

2007 - RX by
Erik Meijer

Microsoft

2012 - RxJava 1 by
Ben Christensen

NETFLIX

Boring definitions

4) Reactives extensions **(Once upon a time)**

2007 - RX by
Erik Meijer

Microsoft

NETFLIX

2012 - RxJava 1 by
Ben Christensen

facebook

Boring definitions

4) Reactives extensions **(Once upon a time)**

2013 - RxJava 2 by Dávid Karnok

2014 – RxKotlin 1 & 2 by Ben Christensen & Thomas Nield

Why RxKotlin ?

What is the

Problem ?

```
new Thread() {
 @Override
 public void run() {
   super.run();

   for (File folder : folders) {
     File[] files = folder.listFiles();

     for (File file : files) {
       if (file.getName().endsWith(".png")) {
         final Bitmap bitmap = getBitmapFromFile(file);
         getActivity().runOnUiThread(new Runnable() {

           @Override
           public void run() {
             imageCollectorView.addImage(bitmap);
           }
         });
       }
     }
   }
 }
}.start();
```

What is the solution?

```
Observable.fromArray(folders)
  .flatMap(file → Observable.fromArray(file.listFiles()))
  .filter(file → file.getName().endsWith(".png"))
  .map(file → getBitmapFromFile(file))
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe((Consumer<Bitmap>) bitmap → imageCollectorView.addImage(bitmap));
```

What is the

Problem ?

```
private void downloadParameters() {
   EmvParameterDownloader emvParameterDownloader = new EmvParameterDownloader(context);
   new TerminalParameterDownloader().download(new Runnable(){
     @Override
     public void run() {
       emvParameterDownloader.downloadICParameters(new Runnable(){
         @Override
         public void run() {
           emvParameterDownloader.downloadPublicKeys();
         }
       });
     }
   });
}
```

What is the solution?

```
private Completable downloadParameters() {
  EmvParameterDownloader emvParameterDownloader = new EmvParameterDownloader(context);
  new TerminalParameterDownloader(context).download())
    .andThen(emvParameterDownloader.downloadICParameters())
    .andThen(emvParameterDownloader.downloadPublicKeys());
}
```

What is the

Problem ?

```
private void login() {
  // find user
  return findUser(name, password, new Runnable(){
    @Override
    public void run() {
      updateLastLoginUser(user);
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
          user → handleLogonUser(user);
        }
      });
    }
  });
}
```

What is the solution?

```
private Completable login() {
  // find user
  return findUser(name, password)
    // update last login user
    .doOnSuccess(user → updateLastLoginUser(user))
    .observeOn(AndroidSchedulers.mainThread())
    .flatMapCompletable(user → handleLogonUser(user));
}
```

In conclusion...

(a) Get rid of the callback hell.

(b) Standard mechanism for error handling

(c) It's a lot simpler than regular threading

(d) Simpler way for async operation

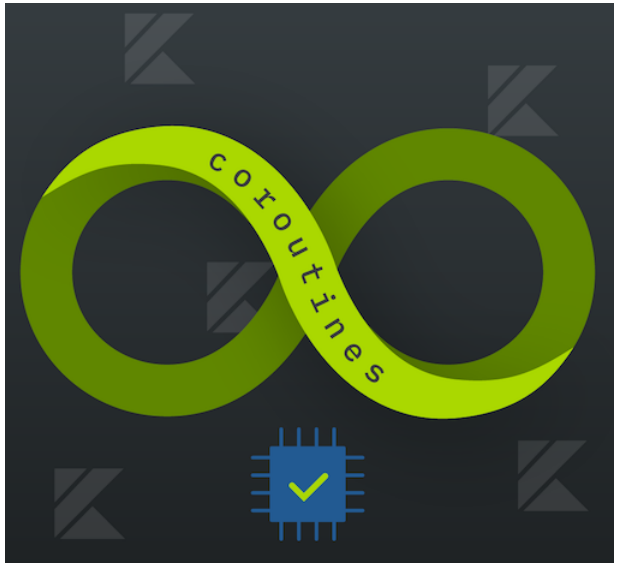
(e) The same api for every operation

(f) The functionnal way

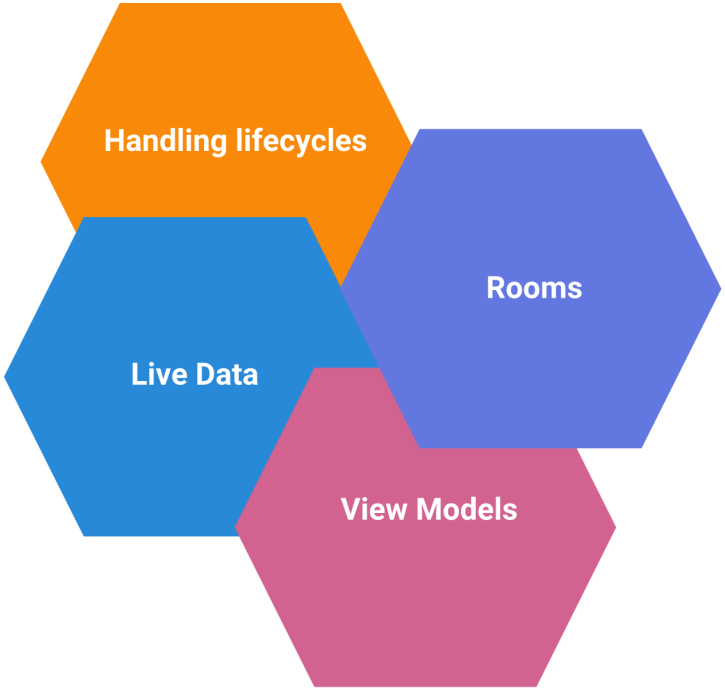
(g) Maintainable & Testable code

(h) Because of Kotlin.

Et Google?



Android Architecture Components



Deep dive into RxKotlin

Pattern Observer

(a) Observable

(b) Observer

(c) Operateurs

(d) Schedulers

(a) Observable

Is a function that when invoked return 0 ~ ∞ values between now and the end of times. Paul Taylor | Reactive 2015

	One value	Many values
Synchronous/ Pull	Fonction	Iterator/ Iterable
Asynchronous/ Push	Promise/ Future	Observable

(a) Observable

Is a function that when invoked return 0 ~ ∞ values between now and the end of times. Paul Taylor | Reactive 2015

	One value	Many values
Synchronous/ Pull	Fonction	Iterator/ Iterable
Asynchronous/ Push	Promise/ Future	Observable

(a) Observable

Is a function that when invoked return 0 ~ ∞ values between now and the end of times. Paul Taylor | Reactive 2015

	One value	Many values
Synchronous/ Pull	Fonction	Iterator/ Iterable
Asynchronous/ Push	Promise/ Future	Observable

(a) Observable

Is a function that when invoked return 0 ~ ∞ values between now and the end of times. Paul Taylor | Reactive 2015

	One value	Many values
Synchronous/ Pull	Fonction	Iterator/ Iterable
Asynchronous/ Push	Promise/ Future	Observable

(b) Observer

It is an interface with 3 main methods.

	Iterable (Pull)	Observer (Push)
Get data	<T> next()	onNext(<T>)
To see the error	<T> next() throws Exception	onError(Throwable)
To complete	boolean hasnext()	onComplete()

(b) Observer

It is an interface with 3 main methods.

	Iterable (Pull)	Observer (Push)
Get data	<T> next()	onNext(<T>)
To see the error	<T> next() throws Exception	onError(Throwable)
To complete	boolean hasnext()	onComplete()

(b) Observer

It is an interface with 3 main methods.

	Iterable (Pull)	Observer (Push)
Get data	<T> next()	onNext(<T>)
To see the error	<T> next() throws Exception	onError(Throwable)
To complete	boolean hasnext()	onComplete()

(c) Type of Observable

Hot Observable

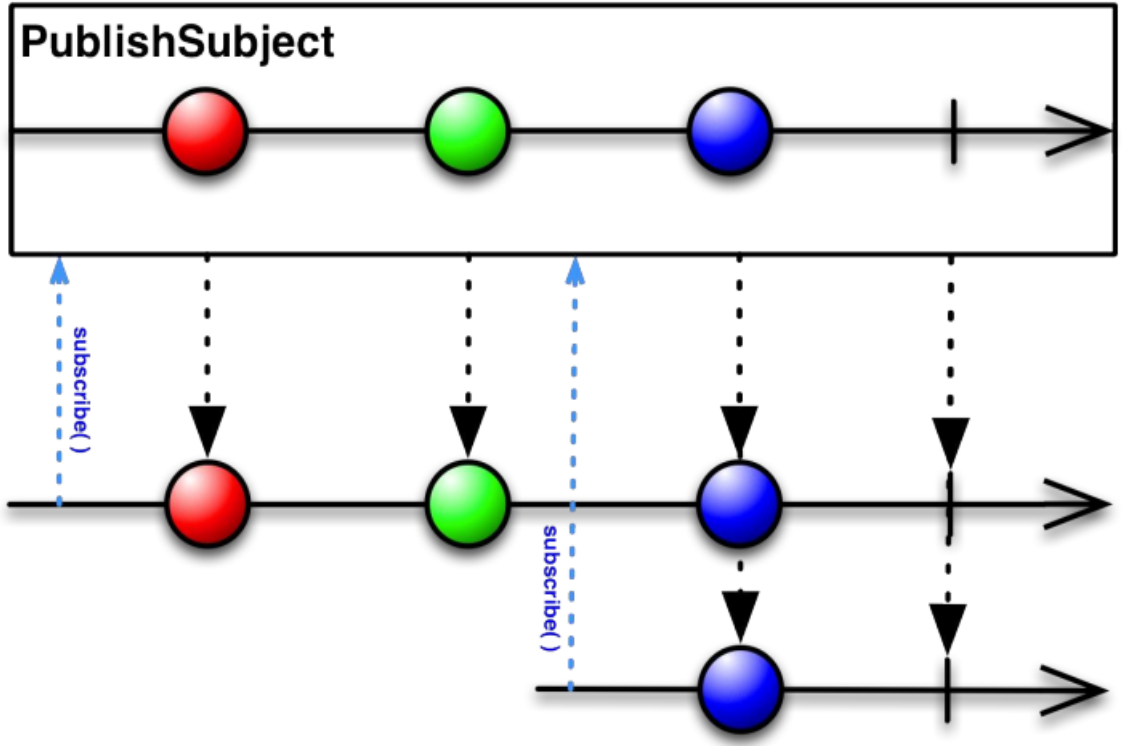
Event no subscriber, always emit events

Cold Observable

No subscriber, no events emit

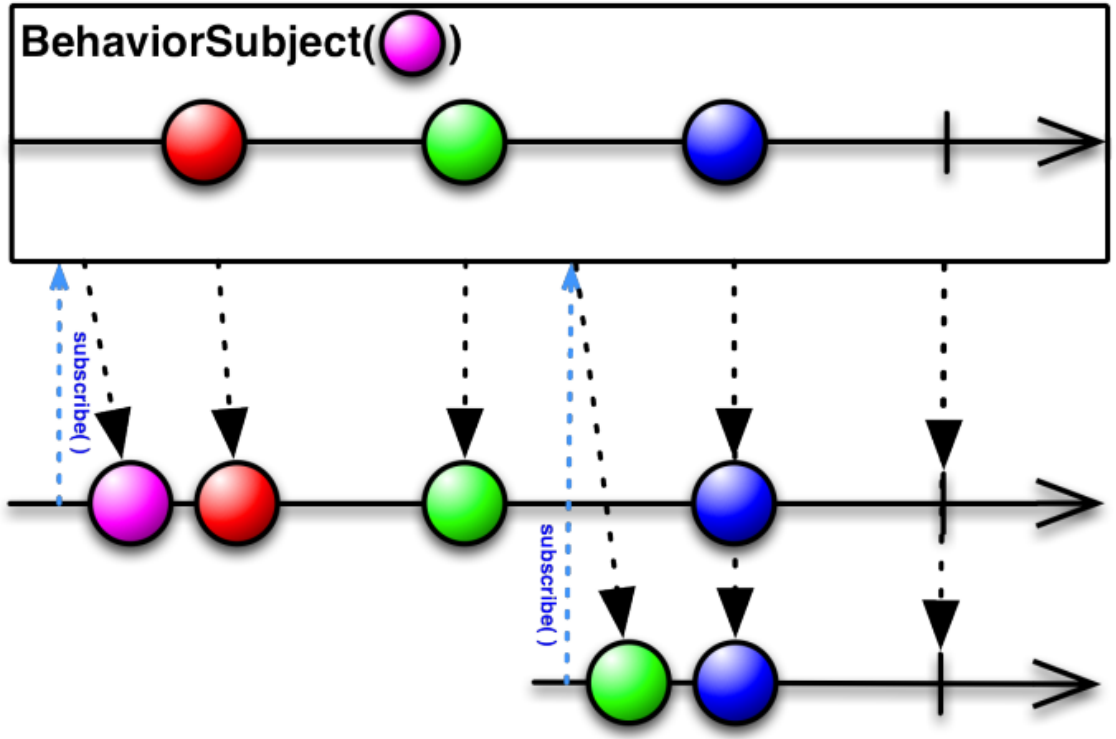
(d) Subjects

Intersection of Observable and observer



(d) Subjects

Intersection of Observable and observer

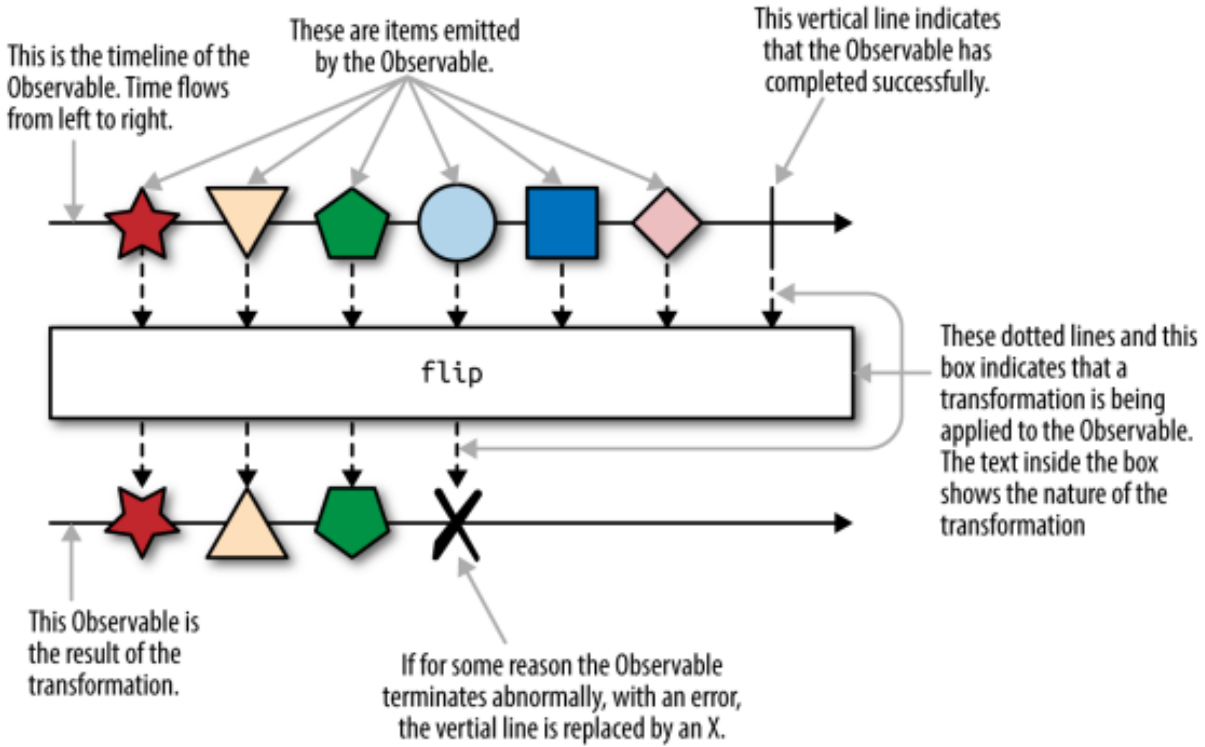


(e) How to emit events

```
import io.reactivex.rxkotlin.subscribeBy
import io.reactivex.rxkotlin.toObservable

fun main(args: Array<String>) {

    val list = listOf("Alpha", "Beta", "Gamma", "Delta", "Epsilon")

    list.toObservable() // extension function for Iterables
            .filter { it.length >= 5 }
            .subscribeBy(  // named arguments for lambda Subscribers
                    onNext = { println(it) },
                    onError =  { it.printStackTrace() },
                    onComplete = { println("Done!") }
            )

}
```

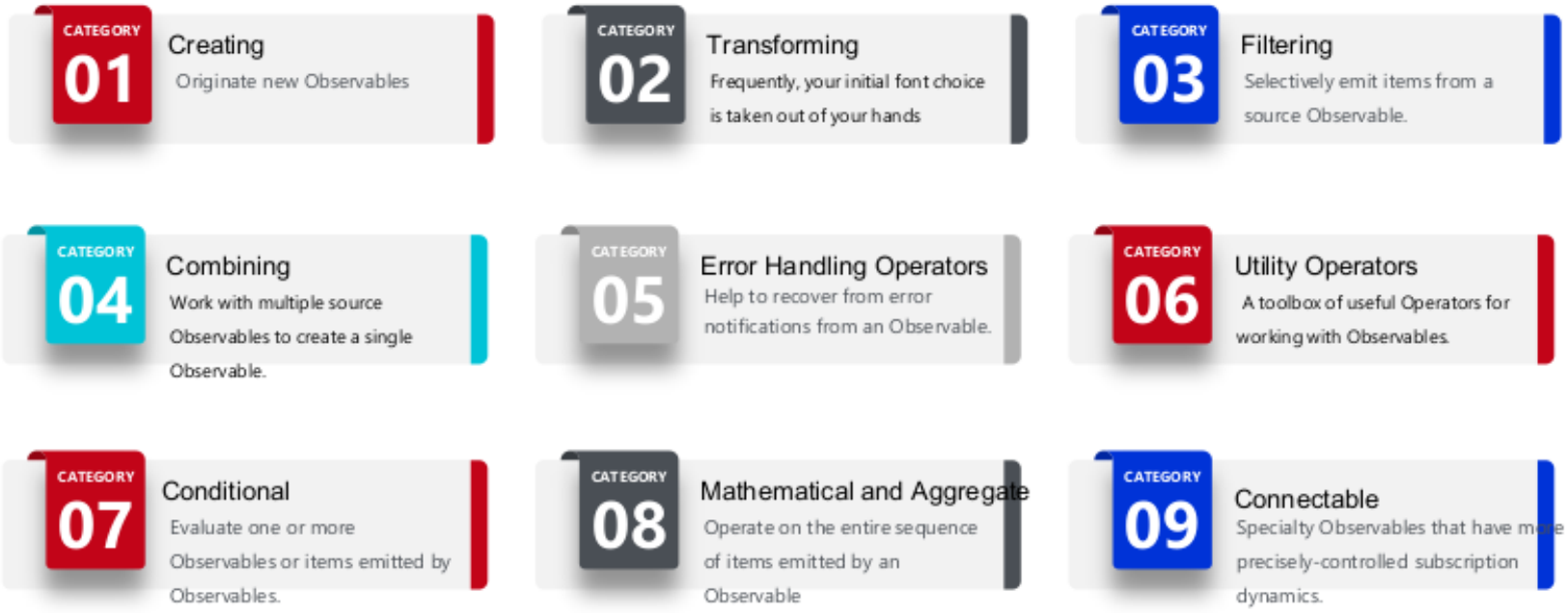
Operators

a) Marble diagrams



Operators

(b) Type of operators



Operators

(c) How to use operators

```
fun simpleComposition() {
    asyncObservable()
        .skip(10)
        .take(5)
        .map { "${it}_xform" }
        .subscribe { println("onNext => $it") }
}
```

Operators

Further exemple

```
interface MavenSearchService {
    @GET("/solrsearch/select?wt=json")
    fun search(@Query("q") s : String, @Query("rows") rows : Int = 20)
        : Observable<MavenSearchResponse>
}

fun main(args: Array<String>) {
    val service = Retrofit.Builder().
            baseUrl("http://search.maven.org").
            addCallAdapterFactory(RxJava2CallAdapterFactory.create()).
            addConverterFactory(MoshiConverterFactory.create()).
            build().
            create(MavenSearchService::class.java)

    service.search("rxkotlin").
            flatMapIterable { it.response.docs }.
            subscribe { artifact ->
                println("${artifact.id} (${artifact.latestVersion})")
            }
}
```

Schedulers

Utiliser pour definir un context d'execution.

- Dans quel thread mon observer va-t-il s'abonner à l'observable ?

  Use subscribeOn() method.

- Dans quel thread mon observer va-t-il observer ?

  Use observeOn() method.

Schedulers

List of schedulers

- Schedulers.newThread()
- Schedulers.io()
- Schedulers.computation()
- Schedulers.trampoline()
- Schedulers.single()
- AndroidSchedulers.mainThread()

Schedulers

Example of schedulers

```
Observable.fromArray(folders)
  .flatMap(file → Observable.fromArray(file.listFiles()))
  .filter(file → file.getName().endsWith(".png"))
  .map(file → getBitmapFromFile(file))
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribe((Consumer<Bitmap>) bitmap → imageCollectorView.addImage(bitmap));
```

Leaning path & Resource

Guide by Arke Team
Reactive Programming with RxJava

- Training / RxJava2-Android-Samples
- kaushikgopal/RxJava-Android-Samples
- Introduction to RxKotlin

Video

- Exploring RxJava 2 for Android
  Jake Wharton
- Intro to RxJava for Android
- Intro to RxJava

Official Site

- ReactiveX Official Site
- ReactiveX GitHub
- RxJava GitHub
- RxKotlin GitHub
- RxJava Category in my Blog

Pratice and Google

- Practice in Project
- Google and Thinking

Conclusion

- Le pattern Observable/Observer peut s'appliquer à tout
- RxJava ou RxKotlin sont des technologies approuvées, déjà utiliser chez NetFlix en production.
- RxJava ou RxKotlin sont des outils très riches.
- Demande une certaine maîtrise, c'est à dire la courbe d'apprentissage est un peu plus longue.

In conclusion...

(a) Get rid of the callback hell.

(b) It's a lot simpler than regular threading

(c) The same api for every operation

(d) Maintainable & Testable code

Fin

Jonathan Monga

Android Developer

JonathanMonga

@jonathan_monga