# RxAndroid

Functional Reactive for Android

Location Labs CoLLoquium
8 August 2015

LOCATION
LABS

# Good Android Applications Today

Responsive UI

Inherently Concurrent

Concurrency hard to get right

How to run async code on Android?

AsyncTasks!

```
264
265     public class DownloadTask extends AsyncTask<String, Void, File> {
266
267         private Callback<File> callback;
268
269         public DownloadTask(Callback<File> callback) {
270             this.callback = callback;
271         }
272
273         // runs in background thread
274         protected File doInBackground(String... args) {
275             final String url = args[0];
276             try {
277                 byte[] fileContent = downloadFile(url);
278                 File file = writeToFile(fileContent);
279                 return file;
280             } catch (Exception e) {
281                 // how to pass error?
282             }
283         }
284
285         // runs on UI thread
286         protected void onPostExecute(File file) {
287             callback.onSuccess(file);
288         }
289     }
290
```

```java
public class DownloadTask extends AsyncTask<String, Void, File> {

    private Callback<File> callback;
    private boolean errorHappened = false;
    private Exception exception;

    public DownloadTask(Callback<File> callback) {
        this.callback = callback;
    }

    // runs in background thread
    protected File doInBackground(String... args) {
        final String url = args[0];
        try {
            byte[] fileContent = downloadFile(url);
            File file = writeToFile(fileContent);
            return file;
        } catch (Exception e) {
            errorHappened = true;
            exception = e;
        }
    }

    // runs on UI thread
    protected void onPostExecute(File file) {
        if (errorHappened) {
            callback.onFailure(e);
        } else {
            callback.onSuccess(file);
        }
    }
}
```

# Execute Tasks on Main Thread

```
352
353     public void onCreate(Bundle savedInstanceState) {
354         super.onCreate(savedInstanceState);
355
356         new MustRunOnMainAsyncTask(new Callback()).execute();
357     }
```

# Callback Hell

```
354
355         new DumbAsyncTask(new Callback() {
356             @Override
357             public void onSuccess() {
358                 new AnotherDumbAsyncTask(new Callback() {
359                     @Override
360                     public void onSuccess() {
361                         new EvenMoreAsyncTask(new Callback() {
362                             // ...
363                         });
364                     }
365                 });
366             }
367         });
368
```

# Activity Lifecycle and Orientation Changes

```
352
353     public void onCreate(Bundle savedInstanceState) {
354         super.onCreate(savedInstanceState);
355
356         new DuplicatingAsyncTask(new Callback()).execute();
357     }
358
```

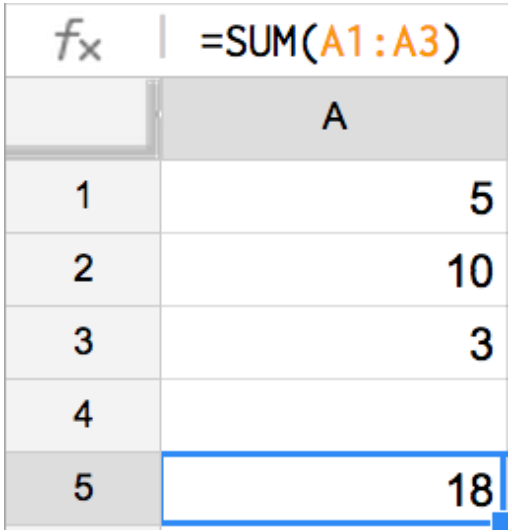
# RxJava / RxAndroid

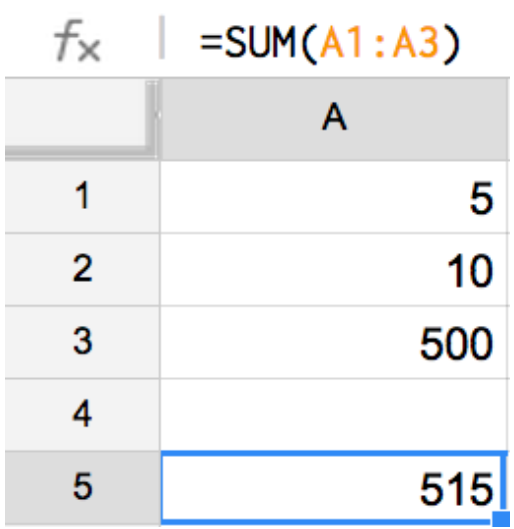
- Functional Reactive Programming
- Observer Pattern Extension

# Imperative Languages

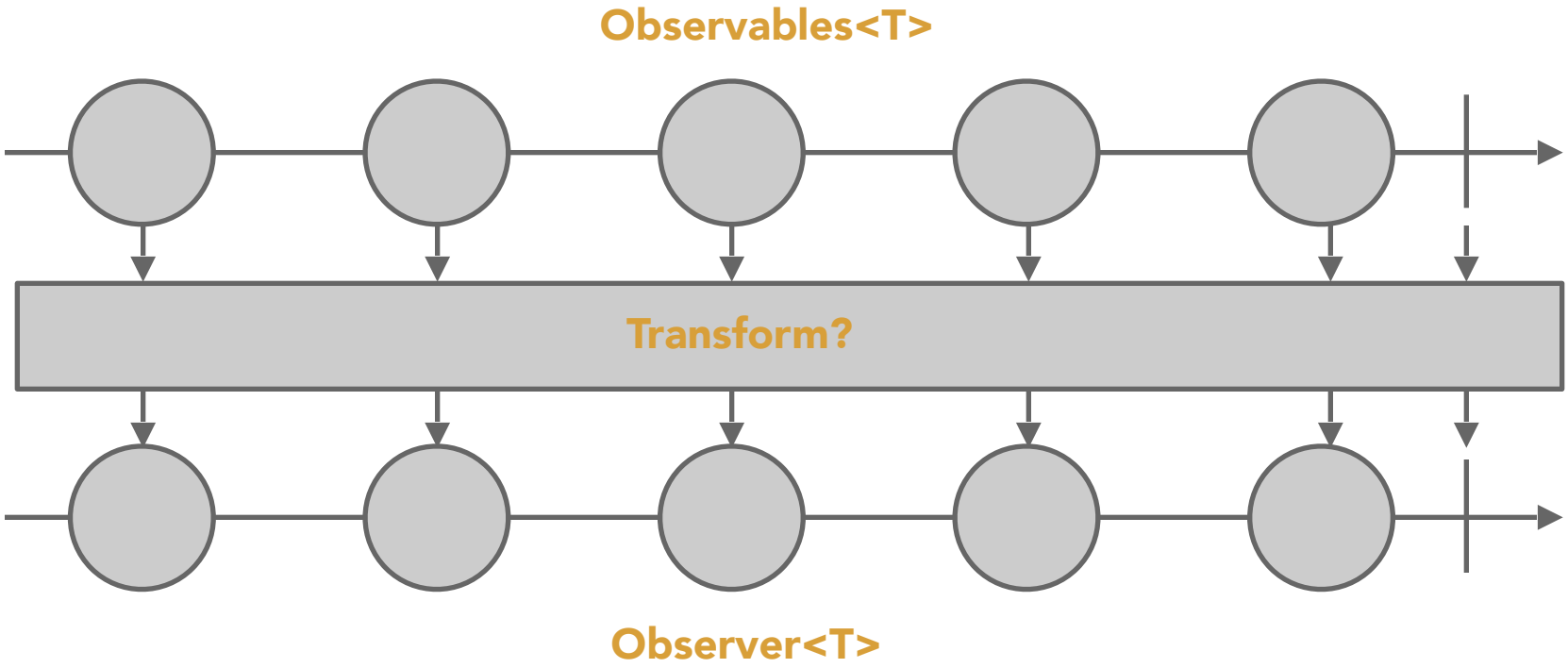
```
a = 0
b = 1
c = a + b
print c // c = 1

a = 5
print c // c = 1
```

# Data Flows!



# RxJava Observer Pattern



# RxJava Big Players

- Observable<T>
- Subscriber<T> extends Observer<T>

# Observer Contract

- onNext(T object)
- onError(Throwable e)
- onComplete()

# What can we do with it?

Reacting to events in a stream:

- Button taps
- Data from a server
- Location updates
- Bluetooth updates
- DB queries
- …

# LL Jira Client

- Create Jira bug tickets from device
- Upload screenshots directly to a ticket
- Predefined application -> project mappings
- Must be on LL wifi or VPN!

# Creating An Observable

```
198
199     public Observable<Project> getProjects() {
200         return Observable.create(subscriber -> {                    // create observable
201             try {
202                 String jsonResponse = jiraApiService.getProjects();
203
204                 JSONArray projects = JSON.getJSONArray(jsonResponse);
205
206                 for (int i = 0; i < projects.length(); i++) {
207                     JSONObject json = (JSONObject) projects.get(i);
208                     subscriber.onNext(new Project(json.toString()));    // onNext()
209                 }
210
211                 subscriber.onCompleted();                           // onCompleted()
212             } catch (Exception e) {
213                 subscriber.onError(e.getCause());                   // onError()
214             }
215         });
216     }
217
```

# RetroLambda - Backport of Java 8 to Java 7

```
35
36          Observable.just("Hello, world!")
37              .subscribe(new Action1<String>() {
38                  @Override
39                  public void call(String s) {
40                      System.out.println(s);
41                  }
42              });
43
```

```
45
46          Observable.just("Hello, world!")
47              .subscribe(s -> System.out.println(s));
48
```

# Creating More Observables!

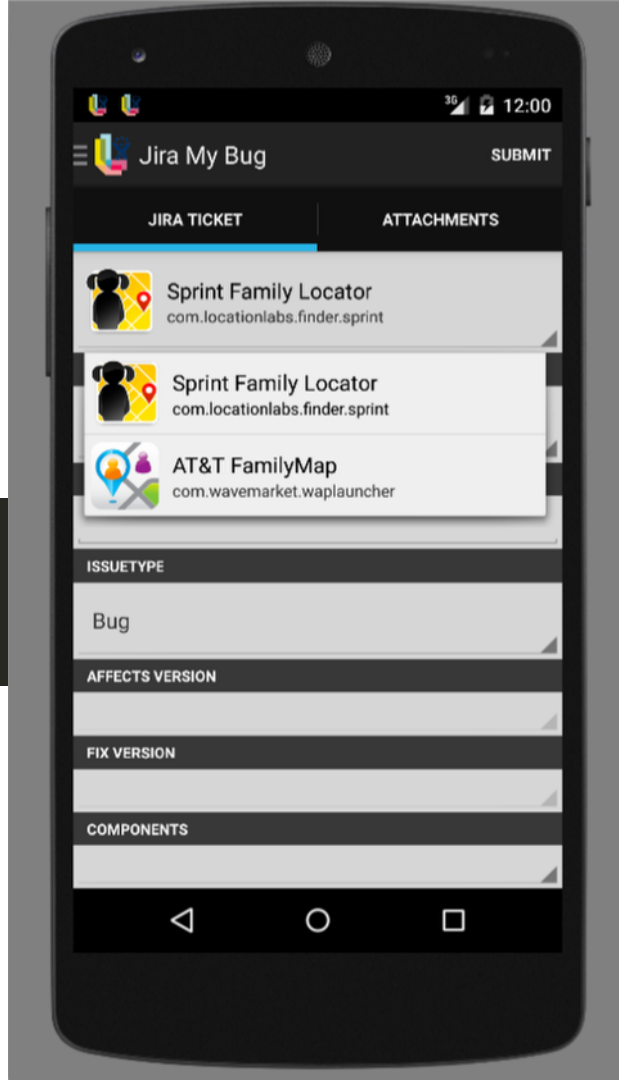
```
88
89      Observable<String> strings = Observable
90          .from(new String[] {"One", "Two", "Three"});
91
92
93      Observable.just("One", "Two", "Three");
94
95
96      Observable.empty();
97
98
99      Observable.error(new Exception());
100
```

# Subscribe

```
70
71      Observable<String> strings = Observable
72          .from(new String[] {"One", "Two", "Three"});
73
74      strings.subscribe(new Observer<String>() {
75          @Override
76          public void onCompleted() {
77              Timber.d("Completed");
78          }
79
80          @Override
81          public void onError(Throwable e) {
82              Timber.d(e.getMessage());
83          }
84
85          @Override
86          public void onNext(String s) {
87              Timber.d(s);
88          }
89      });
90
```

# Operators - Filtering

```
getInstalledApps()
    .filter(app -> app.packageName.contains("locationlabs"))
    .subscribe(app -> appListAdapter.add(app));
```

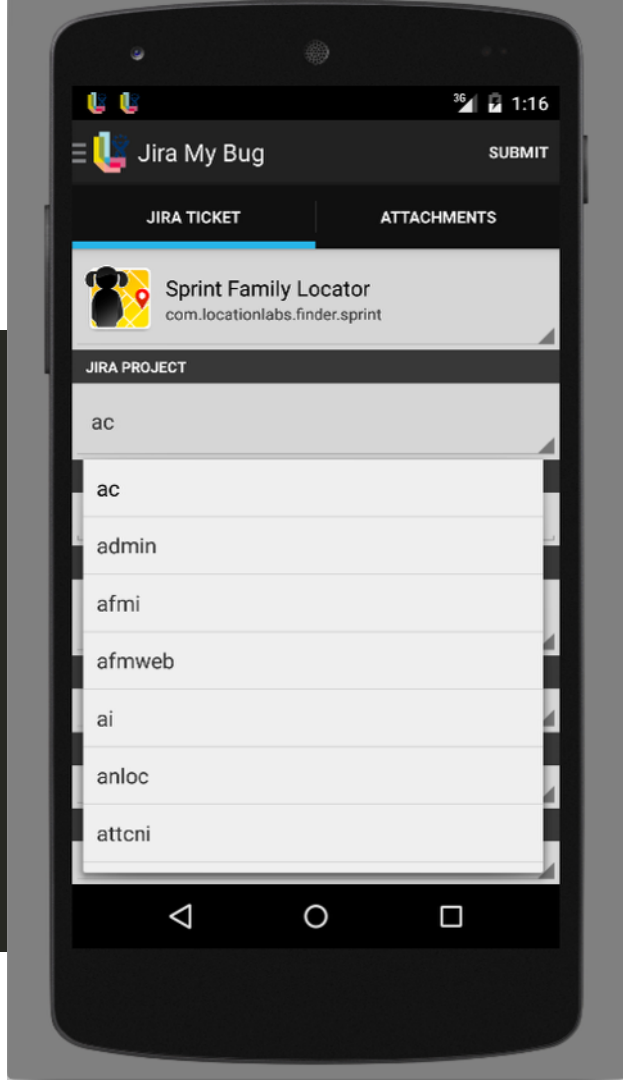


# Operators - More Filtering!

```
228
229         getInstalledApps()
230             .take(5)
231             .subscribe(/**/);
232
233
234         getInstalledApps()
235             .last()
236             .subscribe(/**/);
237
238
239         getInstalledApps()
240             .repeat(2)
241             .distinct()
242             .subscribe(/**/);
243
```

# Transformer Operators - Map

```
getProjects()
    .map((Project p) -> {
        p.setName(p.getName().toLowerCase());
        return p;
    })
    .subscribe(data -> {
        projectAdapter.clear();
        for (Project project : data) {
            projectAdapter.add(project);
        }
    });
```



# Combining Operators - Merge

```
109
110         Observable
111             .merge(
112                 Observable.from(listOfLocalScreenshots),
113                 getScreenshotsFromProject())
114             .subscribe(/**/);
115
```

# Asynchronous + Synchronous - Doesn't Care!

```
117
118         Observable
119             .zip(
120                 getProjectsFromCache(),    // synchronous
121                 getProjectsFromServer(),   // asynchronous
122                 (a, b) -> Arrays.asList(a, b))
123             .subscribe(/**/);
124
```

# Banish Thy Callback Hell - FlatMap!

```
5
6       // get list of projects and print all the issuetypes for each project
7       new GetProjectTask(new Callback<Project>() {
8           public void onSuccess(List<Project> projects) {
9               for (Project project: projects) {
10                  new GetIssueTypes(project.getId(), new Callback<List<IssueType>>() {
11                      public void onSuccess(List<IssueType> types) {
12                          for (IssueType type: types) {
13                              Log.d("Log", type.getName());
14                          }
15                      }
16                  }).execute();
17              }
18          }
19      }).execute();
20
```

```
21
22      getProject(projectName)
23          .flatMap(projects -> Observable.from(projects))                      // emit each project one at a time
24          .flatMap(project -> finderApiService.getIssueTypes(project.getId())) // get and emit issuetypes per project
25          .flatMap(issueTypes -> Observable.from(issueTypes))                  // emit each issuetype one at a time
26          .subscribe(data -> Log.d("IssueType", issueType));                   // subscribe and print!
27
28
29  }
30
```

OK…What about the Android part of RxAndroid?

# ViewObservables/WidgetObservables

```
48
49          ViewObservable
50              .clicks(button)
51              .subscribe(clickEvent -> handleClickEvent());
52
53
54          WidgetObservable
55              .text(searchEditText)
56              .debounce(1, TimeUnit.SECONDS)
57              .subscribe(textChangeEvent -> doSearch(textChangeEvent.text().toString()));
58
```

# AppObservable/LifecycleObservable

```
65
66          AppObservable.bindActivity();
67
68          AppObservable.bindFragment();
69
70          LifecycleObservable.bindActivity();
71
72          LifecycleObservable.bindFragment();
73
```

# Android Schedulers

```
49
50      Subscriber s = new Subscriber<LatLng>() {
51          @Override
52          public void onNext(LatLng latLng) {
53              googleMap.plot(latLng); // update ui
54          }
55      }
56
```

```
61
62      Observable.just(new LatLng(37.8314, -122.2853))
63          .subscribeOn(Schedulers.newThread())          // run observable code
64          .observeOn(AndroidSchedulers.mainThread())  // run subscriber code
65          .subscribe(s)
66
```

# Testing - TestSubscriber

```
94
95      @Test
96      public void testLoadProjects() throw Exception {
97          TestSubscriber<Project> testSubscriber = new TestSubscriber<Project>();
98          jiraService.getProjects().subscribe(testSubscriber);
99          testSubscriber.assertNoErrors();
100         testSubscriber.assertReceivedOnNext(Arrays.asList(project1, project2));
101         testSubscriber.assertCompleted();
102     }
103
```

# Testing Async

```
194
195         Observable.toBlocking( );
196
197         BlockingObservable.from( );
198
199         Observable
200             .from(list)
201             .first()
202             .subscribe(/**/);
203
```

# Conclusion

- Composability
- Powerful operators
- Schedulers
- Solves a lot of problems, but not silver bullet
- People are still figuring things out

# Other Extensions!

- JVM Implementation of Reactive Extensions
- Erik Meijer @ Rx .Net
- Ben Christensen @ RxJava

Remember that 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.

- Ivan Morgillo

# Resources

- https://github.com/ReactiveX/RxJava/wiki/The-RxJava-Android-Module

- http://reactivex.io/

Thanks!

git@git.locationlabs.com:android_jiramybug