RxJava Introduction

Why it is so AWESOME

小创
支付金融

What is RxJava

- Rx-Java

What is Rx

- Reactive eXtension
- Microsoft, Erik Meijer, for .Net
- ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences. It extends the observer pattern to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronisation, thread-safety, concurrent data structures, and non-blocking I/O.
— http://reactivex.io/intro.html

- ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences. It extends the observer pattern to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronisation, thread-safety, concurrent data structures, and non-blocking I/O. — http://reactivex.io/intro.html

Rx and Observer Patter

- The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. — Wikipedia
- ObserverPattern： Subject ——> Observer
- Rx： Observable ——> Observer

What's so special about Rx

- Data manipulation
- Easy Async and Threading
- Easy and safe error handling
- Easily react to Change
- A Functional Programming style
  - No side effect, callback, event

RxJava

- Rx implementation on **JVM**
  - Scala, Closure, Kotlin, JRuby
- iOS: ReactiveCocoa, RxSwift
- JS: RxJs, Beacon
- Rx.Net
- RxRuby
- RxPY

Concepts

- Observable
  - Subject/ Data source in Observer Pattern
  - Emit items
  - “Stream” source
- Observer / Subscriber
  - The “Observer” in Observer Pattern
- Operator
  - Some function for manipulating data, threading, error handling
  - **Core** of Rx
  - The thing you work with most of the time

Observable example

```
Observable<String> observable1 = Observable.just("Hello");

Observable<String> observable2 = Observable.just("Hello", "World");


String[] strArray = new String[]{"Hello", "RxJava", "World"};
Observable<String> observable3 = Observable.from(strArray);

List<String> strList = new ArrayList<>();
for (int i = 0; i < 5; i++) {
    strList.add(i+"");
}
Observable<String> observable4 = Observable.from(strList);


Observable.create(new Observable.OnSubscribe<String>() {
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onStart();
        for (int i=0; i<5; i++) subscriber.onNext("item: "+i);
        subscriber.onCompleted();
    }
});
```

Observer

- Observer
  - onNext, onComplete, onError

```
Observable<String> observable = Observable.just("Hello", "World");
Observer<String> observer = new Observer<String>() {
    @Override
    public void onCompleted() {
        System.out.println("on complete");
    }

    @Override
    public void onError(Throwable e) {
        System.out.println("on error");
        e.printStackTrace();
    }

    @Override
    public void onNext(String s) {
        System.out.println("on next: " + s);
    }
};
observable.subscribe(observer);

//====outputs ================
on next: Hello
on next: World
on complete
```

Ignore onComplete/onError

- Action1 interface

```
Observable<String> observable = Observable.just("Hello", "World");
Action1<String> onNextAction = new Action1<String>() {
    @Override
    public void call(String s) {
        System.out.println(“on next: ”+s);
    }
};
observable.subscribe(onNextAction);
```

```
==========outputs==================
on next: Hello
on next: World
```

Java8 Lambda

```
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        // Run, Forrest, Run!
    }
};
```

Use lambda

```
Runnable runnable = () -> {
    // Run, Forrest, Run!
};
```

If there is only one line

```
Runnable runnable = () -> System.out.println("Run, Forrest, Run!!!");
```

Lambda

```
OnClickListener clickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // View was clicked
    }
};
```

Use lambda

```
OnClickListener clickListener = v -> {
    // View was clicked
};
```

If there is only one line

```
OnClickListener clickListener = v -> doWhateverYouWant(v);
```

Lambdize RxJava

```
Observable<String> observable = Observable.just("Hello", "World");
Action1<String> onNextAction = new Action1<String>() {
    @Override
    public void call(String s) {
        System.out.println(s);
    }
};
observable.subscribe(onNextAction);
```

```
Observable<String> observable = Observable.just("Hello", "World");
Action1<String> onNextAction = s -> System.out.println(s);
observable.subscribe(onNextAction);
```

```
Observable<String> observable = Observable.just("Hello", "World");
observable.subscribe(s -> System.out.println(s));
```

```
==========outputs==================
Hello
World
```

Ignore onComplete

```
Observable<String> observable = Observable.just("Hello", “World");

Action1<Throwable> onErrorAction = new Action1<Throwable>() {
    @Override
    public void call(Throwable e) {
        e.printStackTrace();
    }
};
```

```
observable.subscribe(s -> System.out.println(s), onErrorAction);

observable.subscribe(s -> System.out.println(s), e -> e.printStackTrace());
```

```
==========outputs==================
on next: Hello
on next: World
```

Operators

- Used between Observable and Observer
- Returns another Observable

```
Random random = new Random(System.currentTimeMillis());
List<String> randomNumberStrings = new ArrayList<>();
for (int i = 0; i < 10; i++) {
    randomNumberStrings.add(random.nextInt(10) + "");
}
Observable<String> randomNumberStringObservable = Observable.from(randomNumberStrings);

Observable<Integer> randomNumberObservable = randomNumberStringObservable
                                                    .map(s -> Integer.valueOf(s));
randomNumberObservable.subscribe(integer -> System.out.println(integer));



//More operators
randomNumberStringObservable.map(s -> Integer.valueOf(s))   //Transform String -> Integer
        .filter(integer -> integer > 3)    // take only the ones > 3
        .toSortedList((i1, i2) -> Integer.compare(i2, i1))  //Sort them and return a List
        .flatMap(integerList -> Observable.from(integerList))  //Flattern the list
        .take(2)    // Take the first two
        .subscribe(integer -> System.out.println(integer));
```

More operators

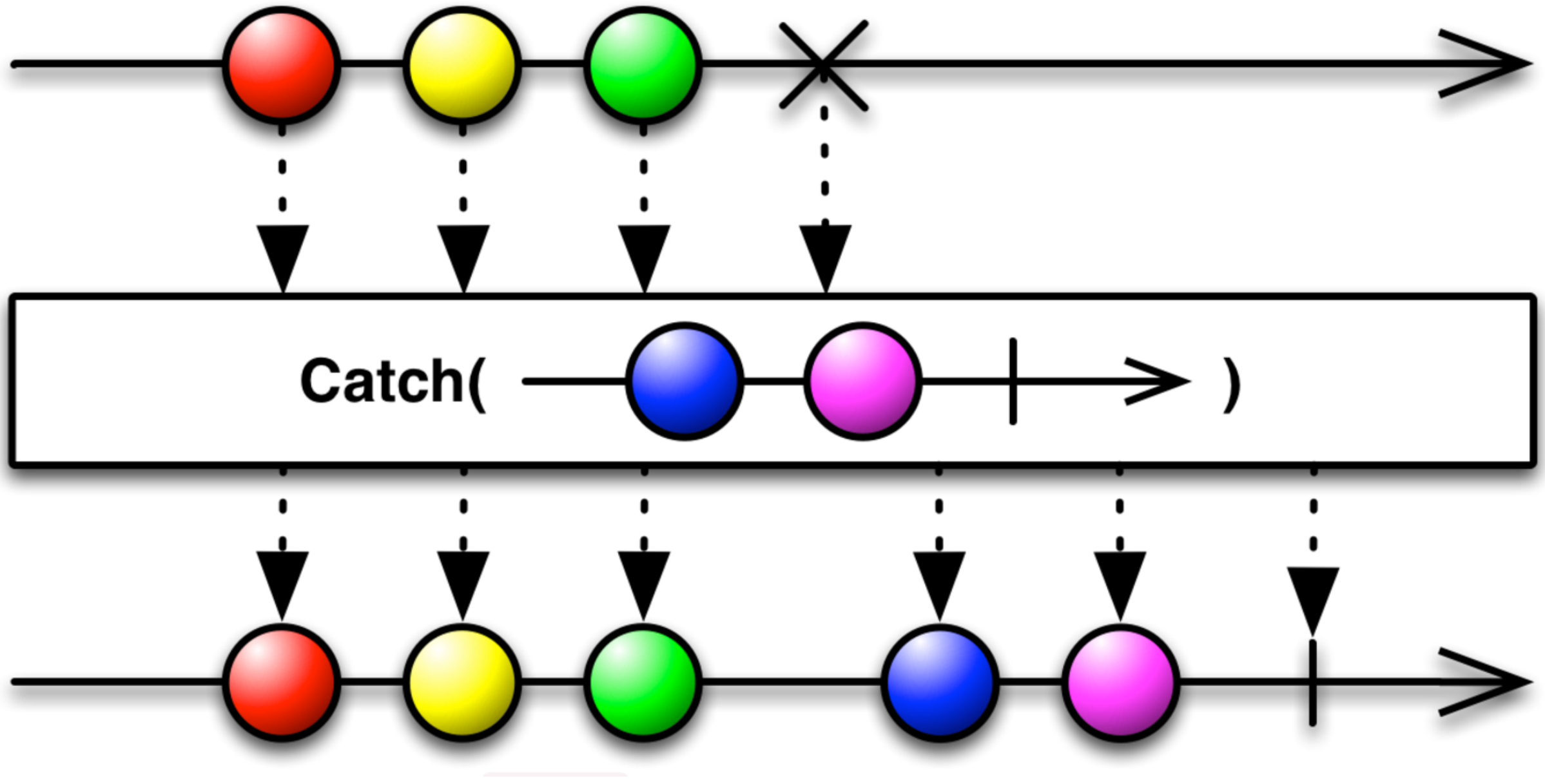
- https://github.com/ReactiveX/RxJava/wiki/Alphabetical-List-of-Observable-Operators
- **Data manipulation**: map, reduce, filter, take, scan, buffer
- **Time manipulation**: delay, throttle, debounce, sample
- **Stream manipulation**: zip, combineLatest, merge, startWith
- **Threading**： subscribeOn， observeOn, toBlock
- **Error handling**： onErrorResumeNext， onErrorReturn, retry， retryWhen

RxMarbles


```
map(x => 10 * x)
```

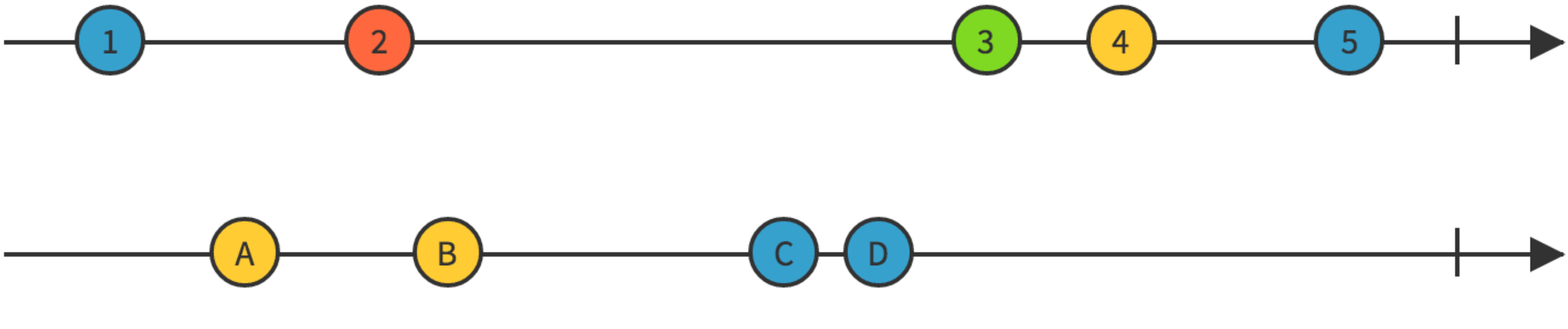
map

RxMarbles



catch/onErrorResumNext

RxMarbles

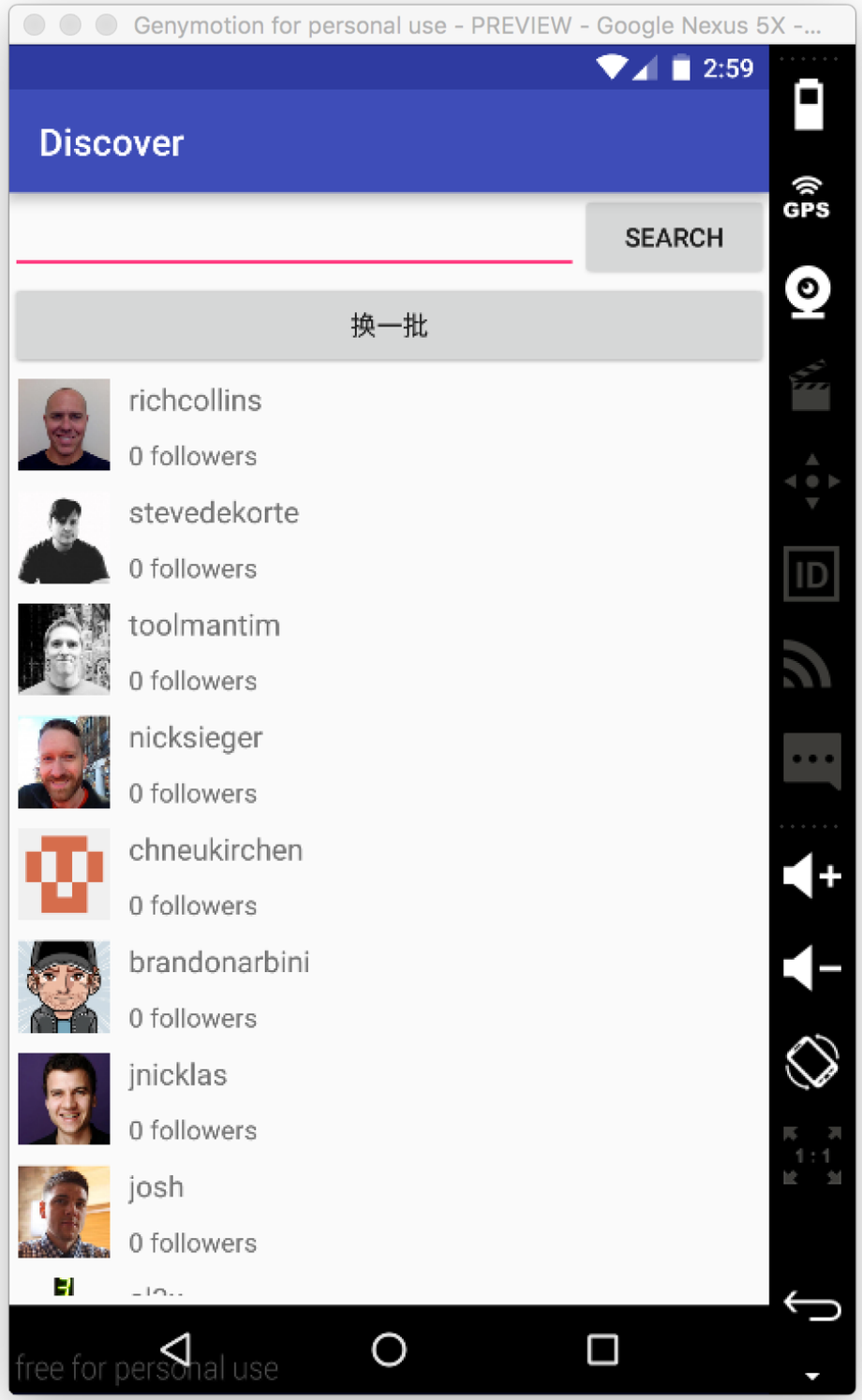



```
combineLatest((x, y) => "" + x + y)
```

combineLatest

Demo App

- Show recommended users on startup
- Refresh
- Search
  - When user is typing
  - When user presses Search button



To start with…

- The brain pattern for programming in Rx

Excuse me?

- What exactly does that mean
- Everything
  - Data
  - Event
  - State
  - …
  - Can be a stream
- Stream???
  - Observable

In a word

- Create an Observable out of everything
  - Data
  - Event
  - State
  - …
- Use Operators to manipulate streams

On Startup

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.another_activity);

    Observable<String> startupStream = Observable.just("on create");

}
```

Get recommended Users

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.another_activity);

    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream
                                                        .map(s -> getRecommendedUsers());
}
```

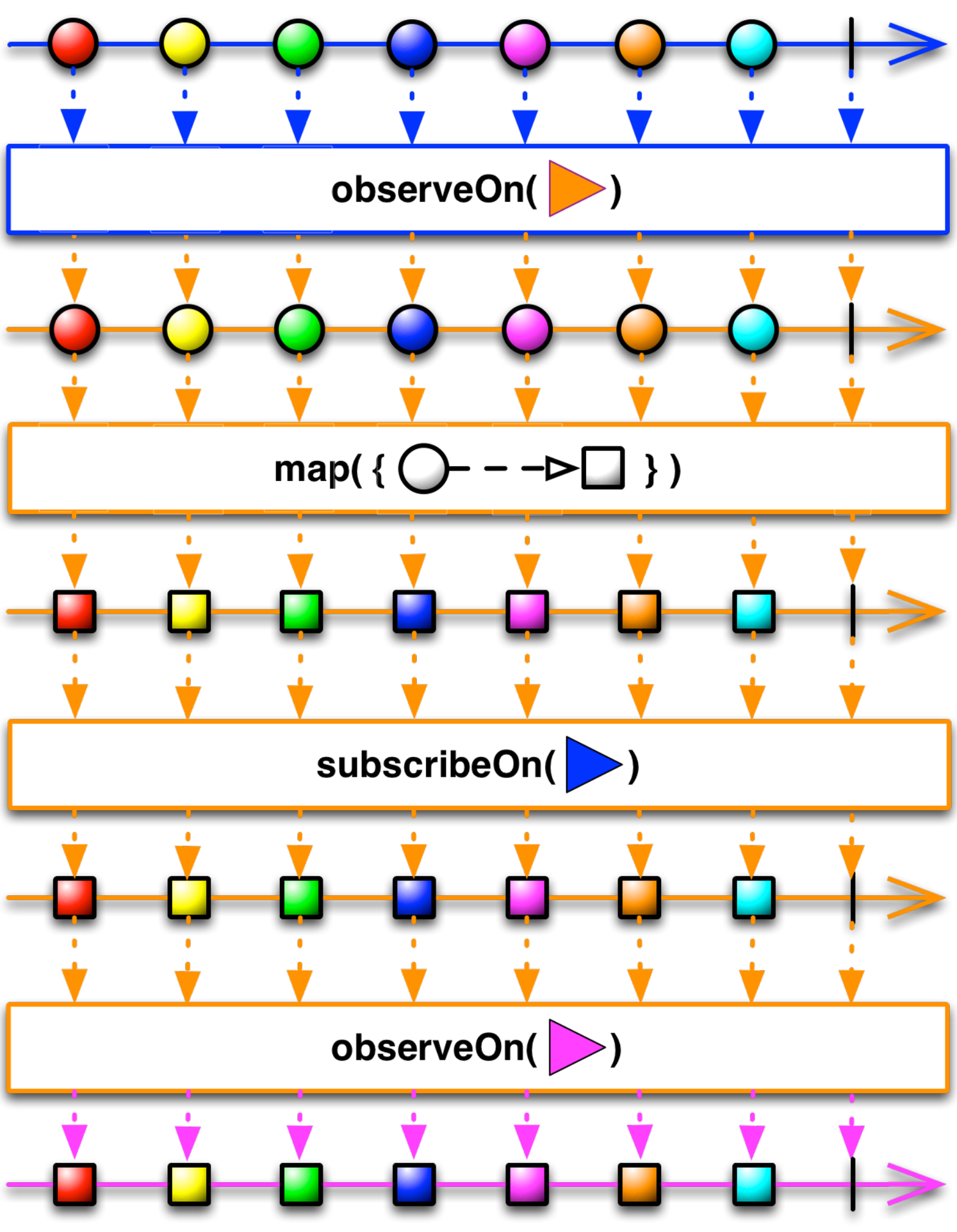
In a background Thread

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.another_activity);

    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream
                                                    //make map run in background
                                                    .map(s -> getRecommendedUsers());

}
```

Threading

- observeOn: specify thread for downstream operators and subscriber
- subscribeOn: specify thread for Observable and following…

```
Observable.from(getDataArray())
        .map(...)
        .filter(...)
        .zipWith(...)
        .observeOn(Schedulers.newThread())  //observeOn在这里
        .toList()
        .flatMap(…)
        .observeOn(Schedulers.computation())    //又一个observeOn在这里
        .take(3)
        .subscribeOn(Schedulers.io())   //subscribOn在这里
        .subscribe(s -> System.out.println(s), e -> e.printStackTrace());
```


observeOn( )
map({ })
subscribeOn( )
observeOn( )

In a background Thread

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.another_activity);

    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> getRecommendedUsers());
}
```

Show them out

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.another_activity);

    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> getRecommendedUsers());

    Subscription subscribe = startupUserStream
                                .observeOn(AndroidSchedulers.mainThread())
                                .subscribe(users -> updateUserList(users),
                                           e -> showError(e));


}
```

Refresh

- Everything is an Observable, remember?
- Make an Observable out of onClick event

```
Observable<Void> clickStream = RxView.clicks(findViewById(R.id.refresh));

Observable<List<User>> clickUserStream = clickStream.observeOn(Schedulers.io())
                                                    .map(ignored -> getRecommendedUsers());
Subscription subscribe1 = clickUserStream.observeOn(AndroidSchedulers.mainThread())
                                         .subscribe(users -> updateUserList(users),
                                                    e -> showError(e));
```

What we have by now

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.discovery_activity);

    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> getRecommendedUsers());
    Subscription subscribe = startupUserStream
                                    .observeOn(AndroidSchedulers.mainThread())
                                    .subscribe(users -> updateUserList(users), e -> showError(e));

    Observable<Void> clickStream = RxView.clicks(findViewById(R.id.refresh));
    Observable<List<User>> clickUserStream = clickStream.observeOn(Schedulers.io())
                                                        .map(ignored -> getRecommendedUsers());
    Subscription subscribe1 = clickUserStream.observeOn(AndroidSchedulers.mainThread())
                                             .subscribe(users -> updateUserList(users),
                                                        e -> showError(e));


}
```

Merge

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.discovery_activity);

    Observable<Void> clickStream = RxView.clicks(findViewById(R.id.refresh))
                                         .map(ignored -> "Refresh button clicked");
    Observable<String> startupStream = Observable.just("on create");
    Observable<List<User>> startupUserStream = startupStream.mergeWith(clickStream)
                                                            .observeOn(Schedulers.io())
                                                            .map(s -> getRecommendedUsers());
    Subscription subscribe = startupUserStream
                                        .observeOn(AndroidSchedulers.mainThread())
                                        .subscribe(users -> updateUserList(users), e -> showError(e));


    Observable<List<User>> clickUserStream = clickStream.observeOn(Schedulers.io())
                                                        .map(ignored -> getRecommendedUsers());
    Subscription subscribe1 = clickUserStream.observeOn(AndroidSchedulers.mainThread())
                                             .subscribe(users -> updateUserList(users),
                                                        e -> showError(e));


}
```

Search

- Perform search when user starts typing
- Model “typing” as a Stream

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<TextViewAfterTextChangeEvent> textChangeStream =
                                                RxTextView.afterTextChangeEvents(searchBox);
```

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                            .map(textChangedEvent -> textChangedEvent.editable().toString());
```

Search for users when typing

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
        .map(textChangedEvent -> textChangedEvent.editable().toString());

Observable<List<User>> searchUserResultStream = textChangeStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));

Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                   .subscribe(users -> updateUserList(users)
                                                                      e -> showError(e));
```

When Press Search button

```
//When press search button
Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                                                   .map(ignored -> searchBox.getText().toString());
searchButtonClickStream.observeOn(Schedulers.io())
                       .map(s -> searchUsers(s))
                       .observeOn(AndroidSchedulers.mainThread())
                       .subscribe(users -> updateUserList(users),
                                  e -> showError(e));
```

What we have for Search

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                    .map(textChangedEvent -> textChangedEvent.editable().toString());
Observable<List<User>> searchUserResultStream = textChangeStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                   .subscribe(users -> updateUserList(users)
                                                              e -> showError(e));

//When press search button
Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                                                   .map(ignored -> searchBox.getText().toString());
searchButtonClickStream.observeOn(Schedulers.io())
                       .map(s -> searchUsers(s))
                       .observeOn(AndroidSchedulers.mainThread())
                       .subscribe(users -> updateUserList(users),
                                  e -> showError(e));
```

Merge Search Request

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                .map(textChangedEvent -> textChangedEvent.editable().toString());
Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                                .map(ignored -> searchBox.getText().toString());
Observable<String> searchUserRequestStream = textChangeStream.mergeWith(searchButtonClickStream);

Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                .observeOn(Schedulers.io())
                                                .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                .subscribe(users -> updateUserList(users),
                                                           e -> showError(e));

//When press search button
Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                                                .map(ignored -> searchBox.getText().toString());
searchButtonClickStream.observeOn(Schedulers.io())
                       .map(s -> searchUsers(s))
                       .observeOn(AndroidSchedulers.mainThread())
                       .subscribe(users -> updateUserList(users),
                                  e -> showError(e));
```

Genymotion for personal use - PREVIEW - Google Nexus 5X -...

5:52

Discover

chris

SEARCH

换一批

chris

0 followers

defunkt

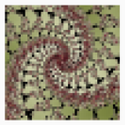
0 followers

webair

0 followers

chrisdickinson

0 followers



colah

0 followers

voodootikigod

0 followers

chrisballinger

0 followers

chriseppstein

0 followers

free for personal use

New requirement

- Perform search only when
  - Input text length >= 3
  - 500 milliseconds after stop typing
  - 相同的字符串不搜索第二遍
  - 如果是手动点的搜索按钮，那么只要又一个字符就要执行搜索

最少3个字符

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> searchButtonClickStream =  RxView.clicks(findViewById(R.id.search_button))
                                                    .map(ignored -> searchBox.getText().toString());
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                    .map(textChangedEvent -> textChangedEvent.editable().toString());

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3);
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                    .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                    .subscribe(users -> updateUserList(users),
                                                                e -> showError(e));
```

停止输入500ms以后

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> searchButtonClickStream =  RxView.clicks(findViewById(R.id.search_button))
                                                    .map(ignored -> searchBox.getText().toString());
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                    .map(textChangedEvent -> textChangedEvent.editable().toString());

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS);
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                    .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                    .subscribe(users -> updateUserList(users),
                                                               e -> showError(e));
```

相同的字符串不在搜索

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> searchButtonClickStream =  RxView.clicks(findViewById(R.id.search_button))
                                                    .map(ignored -> searchBox.getText().toString());
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                    .map(textChangedEvent -> textChangedEvent.editable().toString());

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS)
                                                              .distinctUntilChanged();
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                             .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                .observeOn(Schedulers.io())
                                                .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                .subscribe(users -> updateUserList(users),
                                                           e -> showError(e));
```

Press Search Button

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> searchButtonClickStream =  RxView.clicks(findViewById(R.id.search_button))
                                                    .map(ignored -> searchBox.getText().toString());
                                                    .filter(s -> s.length()>0);
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
                                    .map(textChangedEvent -> textChangedEvent.editable().toString());

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS)
                                                              .distinctUntilChanged();
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                    .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                    .subscribe(users -> updateUserList(users),
                                                               e -> showError(e));
```

Extra requirement

- Show recommended users when text is empty

```
Observable<String> textChangeStream = ……
Observable<String> recommendedUserRequestStream = startupStream.mergeWith(refreshClickStream)
                                .mergeWith(textChangeStream.filter(s -> s.length()==0));
```

What we have now

```
EditText searchBox = (EditText) findViewById(R.id.search_box);

Observable<String> clickStream = RxView.clicks(findViewById(R.id.refresh))
                                       .map(ignored -> "Refresh button clicked");
Observable<String> startupStream = Observable.just("on create");
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
        .map(textChangedEvent -> textChangedEvent.editable().toString());
Observable<String> recommendedUserRequestStream = startupStream.mergeWith(clickStream)
                                .mergeWith(textChangeStream.filter(s -> s.length()==0));
Observable<List<User>> recommenedUserStream = recommendedUserRequestStream.observeOn(Schedulers.io())
                                                                          .map(s -> getRecommendedUsers());
Subscription recommendedUserSub = recommenedUserStream.observeOn(AndroidSchedulers.mainThread())
                                                      .subscribe(users -> updateUserList(users),
                                                                 e -> showError(e));

Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                .map(ignored -> searchBox.getText().toString())
                .filter(s -> s.length()>0);

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS)
                                                              .distinctUntilChanged();
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                               .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.observeOn(AndroidSchedulers.mainThread())
                                                   .subscribe(users -> updateUserList(users),
                                                              e -> showError(e));
```

What we have now

```
EditText searchBox = (EditText) findViewById(R.id.search_box);

Observable<String> clickStream = RxView.clicks(findViewById(R.id.refresh))
                                       .map(ignored -> "Refresh button clicked");
Observable<String> startupStream = Observable.just("on create");
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
        .map(textChangedEvent -> textChangedEvent.editable().toString());
Observable<String> recommendedUserRequestStream = startupStream.mergeWith(clickStream)
                                .mergeWith(textChangeStream.filter(s -> s.length()==0));
Observable<List<User>> recommenedUserStream = recommendedUserRequestStream.observeOn(Schedulers.io())
                                                                          .map(s -> getRecommendedUsers());
Subscription recommendedUserSub = recommenedUserStream.observeOn(AndroidSchedulers.mainThread())
                                                      .subscribe(users -> updateUserList(users),
                                                                 e -> showError(e));

Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                .map(ignored -> searchBox.getText().toString())
                .filter(s -> s.length()>0);

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS)
                                                              .distinctUntilChanged();
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                               .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                    .observeOn(Schedulers.io())
                                                    .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.mergeWith(recommenedUserStream)
                                                   .observeOn(AndroidSchedulers.mainThread())
                                                   .subscribe(users -> updateUserList(users),
                                                              e -> showError(e));
```

Error Handling

One single “Catch”

```
EditText searchBox = (EditText) findViewById(R.id.search_box);
Observable<String> clickStream = RxView.clicks(findViewById(R.id.refresh))
                                       .map(ignored -> "Refresh button clicked");
Observable<String> startupStream = Observable.just("on create");
Observable<String> textChangeStream = RxTextView.afterTextChangeEvents(searchBox)
        .map(textChangedEvent -> textChangedEvent.editable().toString());
Observable<String> recommendedUserRequestStream = startupStream.mergeWith(clickStream)
                            .mergeWith(textChangeStream.filter(s -> s.length()==0));
Observable<List<User>> recommenedUserStream = recommendedUserRequestStream.observeOn(Schedulers.io())
                                                                          .map(s -> getRecommendedUsers());

Observable<String> searchButtonClickStream = RxView.clicks(findViewById(R.id.search_button))
                .map(ignored -> searchBox.getText().toString())
                .filter(s -> s.length()>0);

Observable<String> searchOnTextChangeStream = textChangeStream.filter(s -> s.length() >= 3)
                                                              .debounce(500, TimeUnit.MILLISECONDS)
                                                              .distinctUntilChanged();
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                              .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                            .observeOn(Schedulers.io())
                                                            .map(s -> searchUsers(s));
Subscription searchUserSub = searchUserResultStream.mergeWith(recommenedUserStream)
                                                   .observeOn(AndroidSchedulers.mainThread())
                                                   .subscribe(users -> updateUserList(users),
                                                              e -> showError(e));
```

Caveats

- When error happens, subscriber will be **Unsubscribed**
- No further click event/textChange stream

What can we do?

- map -> flatMap + onErrorReturn

```
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                .observeOn(Schedulers.io())
                                                .map(s -> searchUsers(s));
```

```
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                .observeOn(Schedulers.io())
        .flatMap(s -> searchUsersAndReturnsAnObservable(s).onErrorReturn(new ArrayList<>));
```

Separate “Concerns”

- Separate ongoing stream and one-time stream

```
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                 .observeOn(Schedulers.io())
                                                 .map(s -> searchUsers(s));
```

```
Observable<String> searchUserRequestStream = searchOnTextChangeStream
                                                .mergeWith(searchButtonClickStream);
Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                 .subscribe(s -> performSearch(s));

public void performSearch(String name) {
    userModel.searchUsers(name).map(...)
            .toSortedList(...)
            .flatMap(...)
            .whateverOperator(...)
            .subscribeOn(…)
            .observeOn(...)
            .subscribe(users -> ..., e -> showError(e));
}
```

Custom SuppressErrorOperator

```
public final class OperatorSuppressError<T> implements Operator<T, T> {
    final Action1<Throwable> onError;

    public OperatorSuppressError(Action1<Throwable> onError) {
        this.onError = onError;
    }

    @Override
    public Subscriber<? super T> call(final Subscriber<? super T> t1) {
        return new Subscriber<T>(t1) {

            @Override
            public void onNext(T t) {
                t1.onNext(t);
            }

            @Override
            public void onError(Throwable e) {
                onError.call(e);
            }

            @Override
            public void onCompleted() {
                t1.onCompleted();
            }

        };
    }
}

Observable<List<User>> searchUserResultStream = searchUserRequestStream
                                                .observeOn(Schedulers.io())
                                                .map(s -> searchUsers(s))
                                                .lift(new OperatorSuppressError<>(e -> e.printStackTrace()));
```

Error Handling Operators

- onErrorResumeNext
- onErrorReturn
- retry

Recap: What Rx Gives Us

- Data manipulation
- Easy Async and Threading
- Easy and safe error handling
- Easily react to Change
- A Functional Programming style
  - No side effect, states, callback, event

Resources

- The introduction to Reactive Programming you've been missing -- André Staltz
- Grokking RxJava -- Dan Lew
- Functional Reactive Programming with RxJava -- Ben Christensen
- RxMarbles
- https://www.youtube.com/watch?v=NVKmyK6sd-Q

- Code in this Presentation

Happy Rxing

THANK YOU!