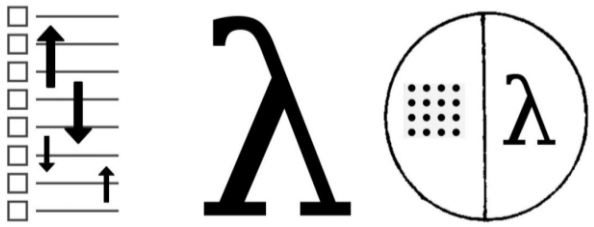
Fundusze Europejskie
Wiedza Edukacja Rozwój

Politechnika Wrocławska

Unia Europejska
Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”



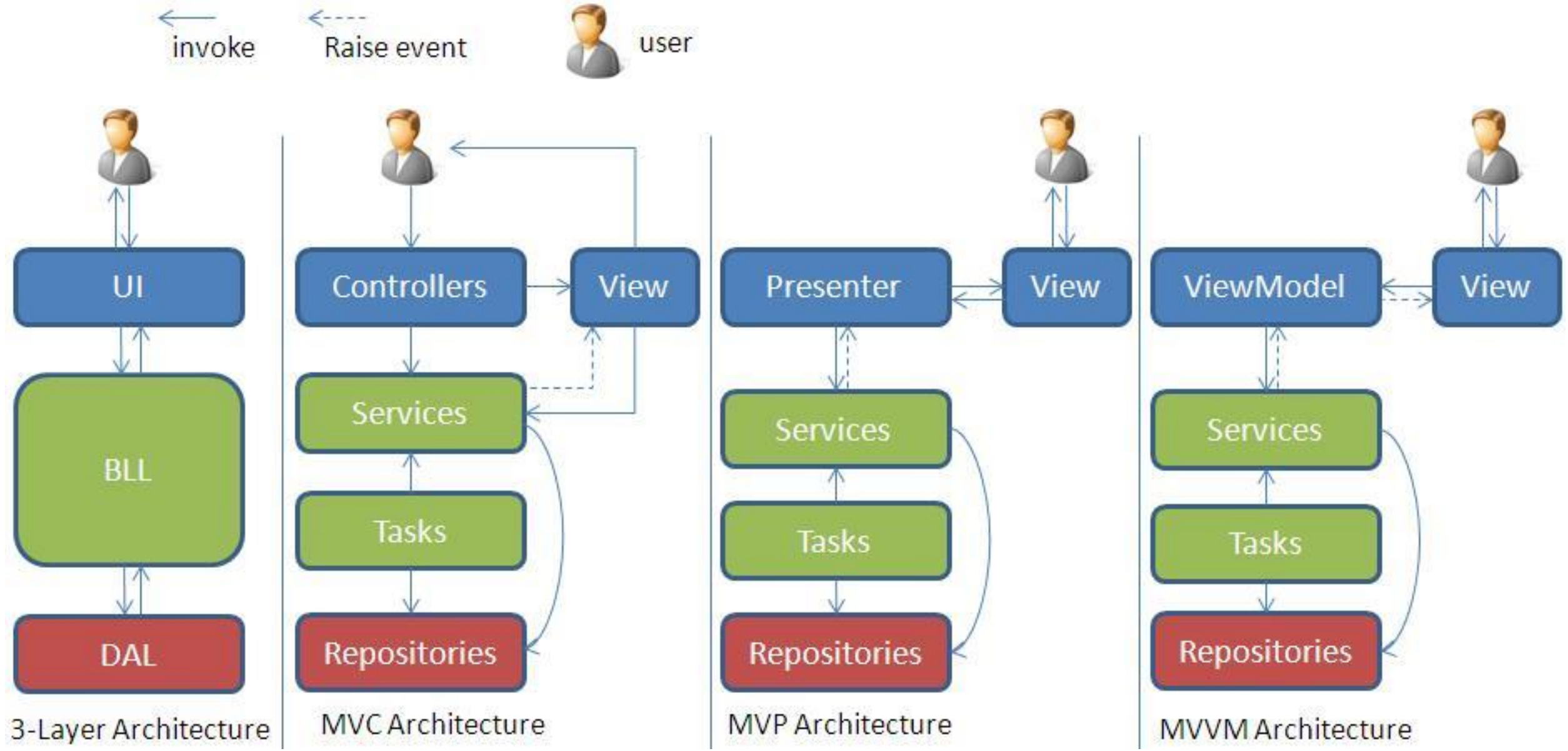
Programming paradigms

L14 part 2: Reactive Programming

by Michał Szczepanik

MV…

- MVP
- MVC
- MVVM
- CLEAN ARCHITECTURE


invoke
Raise event
user
UI
BLL
DAL
3-Layer Architecture
Controllers
View
Services
Tasks
Repositories
MVC Architecture
Presenter
View
Services
Tasks
Repositories
MVP Architecture
ViewModel
View
Services
Tasks
Repositories
MVVM Architecture


RxJava

History of Rx

- Developed by Microsoft as Reactive Extensions
- “...is a library to compose asynchronous and event-based programs using observable collections and LINQ-style query operators.”
- Has been ported on most platforms and languages

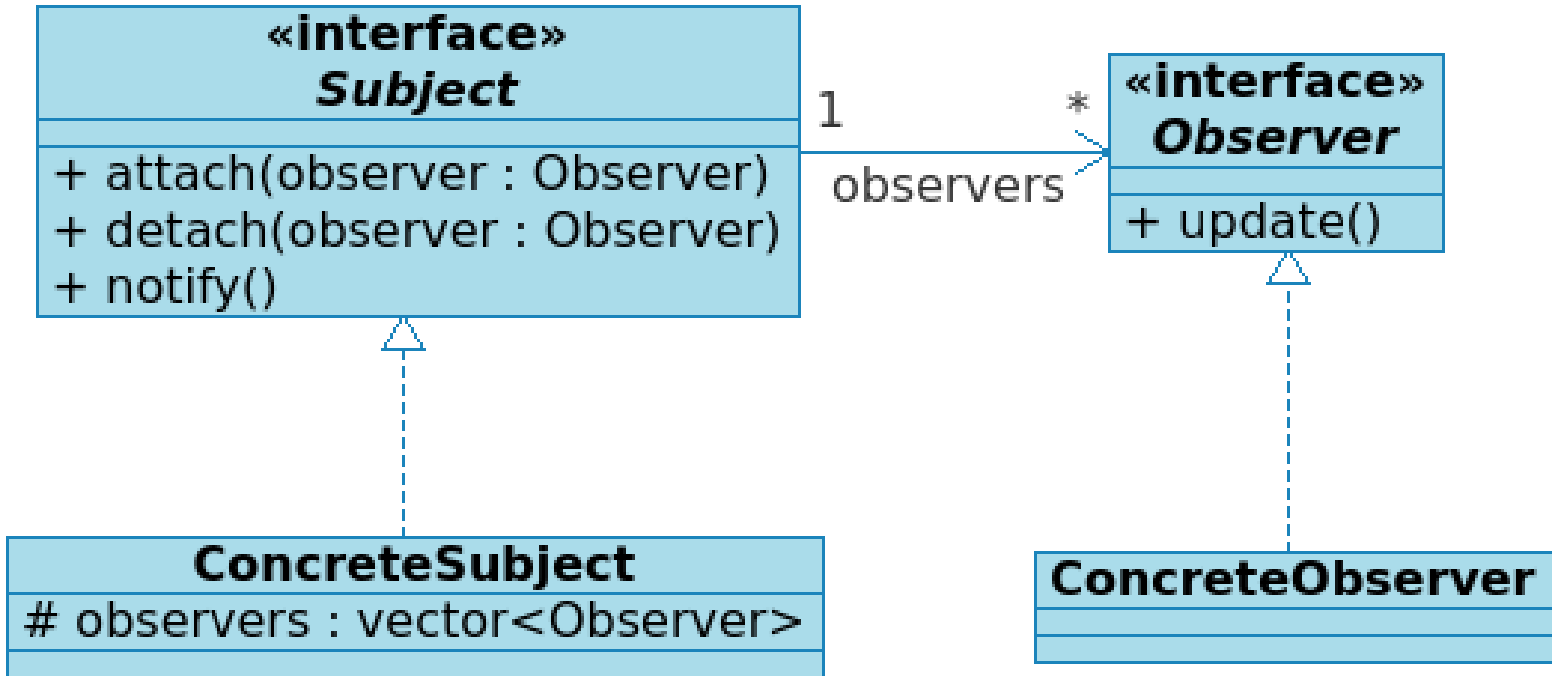
Rx is..

- Known as Functional Reactive Programming
- A way of composing (pure) functions into processing chains
- **A way to avoid callback hell**
- A very fancy event publish / listen mechanism

RxJava 1 vs 2

RxJava 2.0 has been completely rewritten from scratch on top of the Reactive-Streams specification.

Observer pattern



Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure

```
interface Publisher<T> {
        void subscribe(Subscriber<? Super T> s);
}

interface Subscriber {
    void onNext(T t);
    void onComplete();
    void onError(Throwable t);
    void onSubscribe(Subscription s);
}
interface Subscription {
void request(long n);
void cancel();
}

interface Processor<T, R> extends Subscriber<T>, Publisher<R> { }
```

Observable and Flowable

- Observable
  - Emits 0 to n items.
  - Terminates with complete or error.
  - Does not have backpressure.
- Flowable
  - Emits 0 to n items.
  - Terminates with complete or error.
  - Has backpressure.

Back... what ? Backpressure

Backpressure allows you to control how fast a source emits items.
(RxJava 1.x added backpressure late in the design process.)

Observable and Flowable

```
interface Observer {
        void onNext(T t);
        void onComplete();
        void onError(Throwable t);
        void onSubscribe(Disposable d);
}

interface Disposable {
        void dispose();
}
```

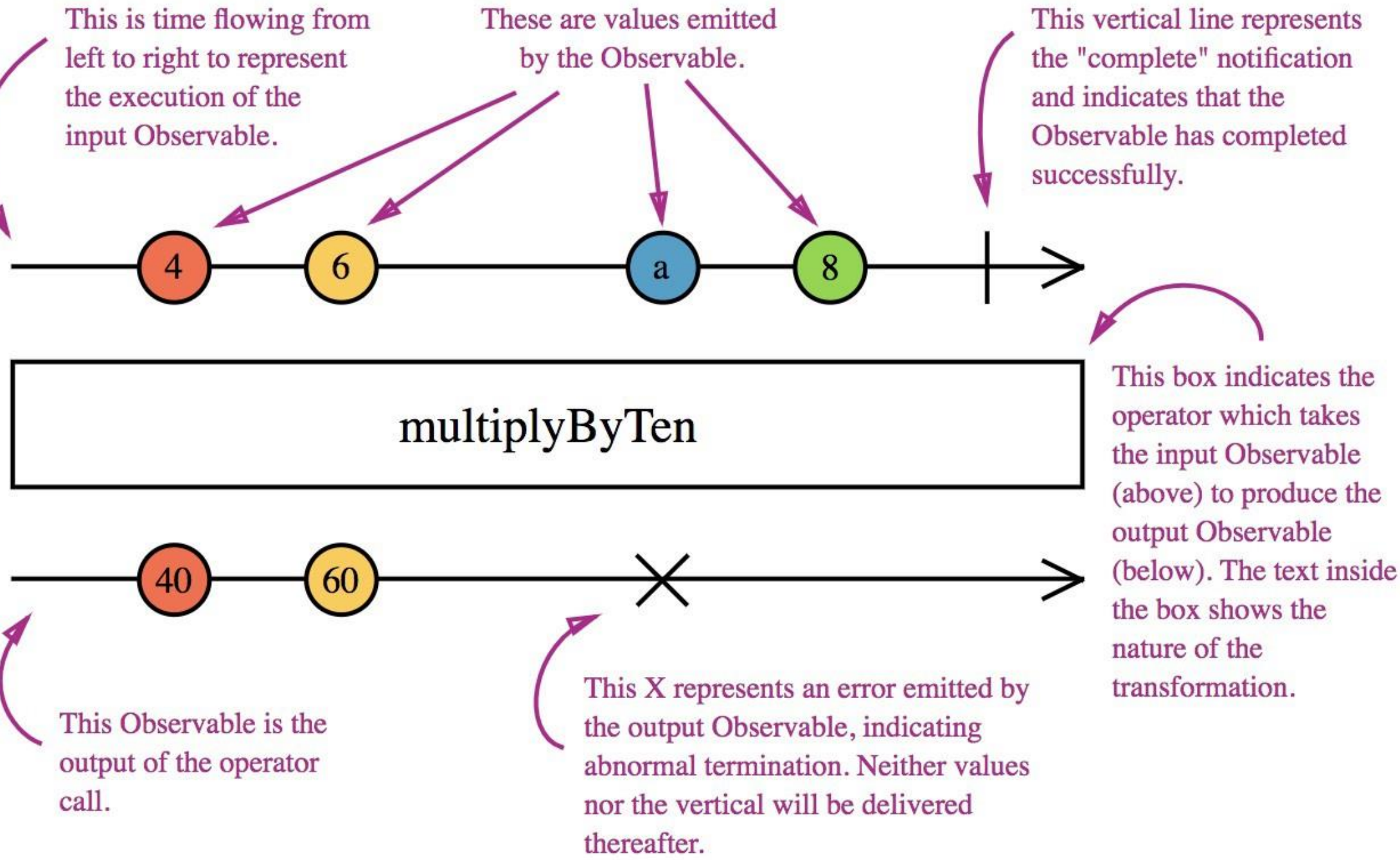
```
interface Subscriber {
        void onNext(T t);
        void onComplete();
        void onError(Throwable t);
        void onSubscribe(Subscription s);
}

interface Subscription {
        void cancel();
        void request(long r);
}
```

Operators

Operators do three things:

- Manipulate or combine data in some way.
- Manipulate threading in some way.
- Manipulate emissions in some way.

```
Observable<String> greeting = Observable.just("Hello");
Observable<String> yelling = greeting.map(s -> s.toUppercase());
```


This is time flowing from left to right to represent the execution of the input Observable.
These are values emitted by the Observable.
This vertical line represents the "complete" notification and indicates that the Observable has completed successfully.
4
6
a
8
multiplyByTen
This box indicates the operator which takes the input Observable (above) to produce the output Observable (below). The text inside the box shows the nature of the transformation.
40
60
This Observable is the output of the operator call.
This X represents an error emitted by the output Observable, indicating abnormal termination. Neither values nor the vertical will be delivered thereafter.

More:

- Doc:

http://reactivex.io/

- Samples:

https://github.com/kaushikgopal/RxJava-Android-Samples

- Diagrams:

https://rxmarbles.com/

What can I use Rx for?

- Network calls
- Async/Sync calls
- Views
- Event listeners
- Creating new objects from existing data
- And so much more!

Thank you for your attention