RxJava

- can be though as **library** that give us easy way to use **observer pattern.**
- there are Java Observale but it is **complex**, **layered**.
- is a Java implementation for **ReactiveX**.

Component

Observable	**Observer(Subscribers)**
what you watch	watcher
list of values/data	consumer for data

Observable Categories

- New event: (UI events).
- Change event: (change title after data update).
- Complete event: tasks of code (network calls).

Subscriber methods

- `onSubscribe()`-> when subcribe to observable
- `onNext()`-> when a new value is emited/publisged
- `onError()` -> when error ocurr
- `onComplete()` -> when complete task

Observable Types

- **Relay**
    - easiest piece of work.
    - **hot observable**: only get **most current values** or default
    - **never error out** or **complete** so it is **save for UI**.
- **Subject**
    - little more complex.
    - can **receive onComplete/onError** so *not ideal* for UI.
    - **Die** after onComplete/onError.
    - **can observe** and also **be observed**.
    - **hot observable**:but it **receive number of events** depending of its type.
    - flavor of subjects:
        - **Behavior**:- **last event** or defaut (ex as kid remember last thing).
        - **Publish** :- start from **zero events** so use **only new events**(ex for academia no matter books published by prof only concern what new book to publish).

    - **Replay**:- hav n of previous events `let subject = ReplaySubject<String>.create(bufferSize : 3)`

- **Observable**: used for **complex tasks** such as chaining network calls.

Traits

are one-off tasks that cann be wrapped in asingle observable, there are

- **Single** that accept (onNext, onError).
- **Completable** that accept (onComplete, onError).
- **Maybe** that accept (onNext/onComplete, onError).

when you see **one-off call** such as **network call** so it is a greate place to **use Traits**.

Relay Example.

- Relay: only default or last value
- it is set and get.
- not accept onError/onComplete.

Code

```
val behavoiur = BehaviorRelay.createDefault("1")
println("~~~ Relay ~~~ ")
println(" 1 value ${behavoiur.value}")
behavoiur.accept("10")
println(" 2 value ${behavoiur.value}")
behavoiur.accept("12") // is ignored/overwritten
behavoiur.accept("13")
println(" 3 value ${behavoiur.value}")
```

Output

```
~~~ Relay ~~~
1 value 1
2 value 10
3 value 13
```

`12` is not shown as Relay just use last value only.

Subjects Example

Code

```
    var behaviorSubject = BehaviorSubject.createDefault("10")
    // disposable hold reference to subscription
    var disposable = behaviorSubject.subscribe(
            { newValue -> println("behaviorSubject new Value $newValue")},
            { error-> println("behaviorSubject error ${error.localizedMessage}")}
,
            { println("behaviorSubject complete")} ,
            { dispose ->
            // called with each new subscribtion
                println("behaviorSubject subscribed")
            } )


    behaviorSubject.onNext("1") // publish values
    behaviorSubject.onNext("2")
    behaviorSubject.onNext("2")

    behaviorSubject.onComplete() // subject dies after onComplete
    behaviorSubject.onNext("3") // not shown
```

Output

```
I/System.out: behaviorSubject subscribed
I/System.out: behaviorSubject new Value 10
I/System.out: behaviorSubject new Value 1
behaviorSubject new Value 2
behaviorSubject new Value 2
I/System.out: behaviorSubject complete
```

#Observable

Code

```
var observable = Observable.create<String>{
    observer -> // ref for new observer
    println("triggered with every subscriber.")
    // do your work in background thread
    launch {
        delay(1000)
        observer.onNext("1") // publish data to observer
        observer.onNext("2")
        observer.onComplete() // finish publishing
    }
```

```
}

observable.subscribe {
        newVal -> // onNext
        println("new val $newVal")
}.disposedBy(bg) // to release resources

observable.subscribe {
        newVal -> // onNext
        println("second new val $newVal")
}.disposedBy(bg) // to release resources
```

Output

```
I/System.out: triggered with every subscriber. // first subscriber
I/System.out: triggered with every subscriber. // second subscriber
I/System.out: new val 1
I/System.out: second new val 1
I/System.out: new val 2
I/System.out: second new val 2
```

- First **two lines for subscribtions**
- Then because of `observer.onNext("1")` so values are **emitted** to subscribers so output as third and fourth line.
- And same occured with `observer.onNext("2")`.

Traits

more specialized observables. // add comp. later

```
// Single
/*
* Single used when only single value is returned
* has onSuccess (i.e same as onNext), onError*/

val single = Single.create<String> {
    val success = false
    if (success) {
        it.onSuccess("Work done")
    } else {
        val error = IllegalAccessException("Fake error")
        it.onError(error)
    }
}

single.subscribe({ value -> println("single -> $value") },
```

```
        { error -> println("single error $error") }).disposedBy(bg)


// Completeable
val completable = Completable.create {
    // do logic here
    val success = false
    if (success) {
        it.onComplete() // indicate that task is done successfully
    } else {
        val error = IllegalAccessException("Fake error")
        it.onError(error) // error with doing Task
    }
}

completable.subscribe({ println("Complete ")} ,
            {error -> println("error $error" )}).disposedBy(bg)

// Maybe
val maybe = Maybe.create<String> {
    val success = true
    val hasValue = false
    if (success){
        if (hasValue){
            it.onSuccess("new value ")
        }else{
            it.onComplete()
        }
    }else{
        val error = IllegalAccessException("Fake error")
        it.onError(error) // error with doing Task
    }
}

maybe.subscribe({ value -> println("May be value ")} ,
        {error -> println ("Maybe error $error")} ,
        {
            println("Complete Maybe")
        }).disposedBy(bg)

}
```

Output

```
I/System.out: single error java.lang.IllegalAccessException: Fake error
I/System.out: error java.lang.IllegalAccessException: Fake error
I/System.out: Complete Maybe
```