RxSwift Cheatsheet

SUBJECT TYPES

	Subject	Relay
Publish	No state, errors	No state, no errors
Behavior	State, errors	State, no errors

UIKIT EXTENSIONS

```
let button: UIButton = UIButton(frame: .zero)
button.rx.tap // ControlEvent<Void>

let control: UIControl = UIControl(frame: .zero)
control.rx.controlEvent(_ controlEvents: UIControl.Event) // ControlEvent<Void>

let segmentedControl: UISegmentedControl = UISegmentedControl(frame: .zero)
segmentedControl.rx.selectedSegmentIndex // ControlProperty<Int>

let textField: UITextField = UITextField(frame: .zero)
textField.rx.text // ControlProperty<String?>

let tableView: UITableView = UITableView(frame: .zero, style: .plain)
tableView.rx.itemSelected // ControlEvent<IndexPath>
tableView.rx.modelSelected(_ modelType: T.Type) // ControlEvent<T>
tableView.rx.items(cellIdentifier: String) // (O) -> (@escaping (Int, S.Iterator.Element, Cell) -> Void) ->
Disposable where S : Sequence, S == O.E, Cell : UITableViewCell, O : ObservableType

let notificationCenter: NotificationCenter = NotificationCenter.default
notificationCenter.rx.notification(_ name: Notification.Name?) // Observable<Notification>
```

MORE OPERATORS

- List: http://reactivex.io/documentation/operators.html
- Visual aid: http://rxmarbles.com

RxSwift Cheatsheet

BASIC INFORMATIONS

Observable sequences can emit 0 or more events over their lifetimes. 3 types of events:

- .next(value: T) – new value from the observable
- .error(error: Error) – observable encountered the error; terminates the sequence
- .completed – observable finished emitting events; terminates the sequence

CREATING OBSERVABLES

asObservable - convert various objects into Observables

```
let field = UITextField(frame: .zero)
field.rx.text.orEmpty.asObservable()
```

create - create an Observable from scratch

```
Observable<String>.create { subscribe in
    subscribe.onNext("Rx Workshops are cool")
    subscribe.onCompleted()

    return Disposables.create()
}
```

from(_: [T]) - convert an array into Observable

```
Observable<Int>.from([1, 2, 3])
    .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

of - convert values into Observable

```
Observable<Int>.of(1, 2, 3)
    .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

empty - emit no items; terminates normally

error - emit no items; terminates with an error

```
Observable<String>.error(RxError.unknown)
    .subscribe { print($0) }
```

```
error(Unknown error occurred.)
```

never - emit no items; does not terminate

just - emit a particular item

```
Observable<String>.just("Hello Rx!")
    .subscribe { print($0) }
```

```
next(Hello Rx!)
completed
```

TRANSFORMING OBSERVABLES

map

```
Observable<Int>.of(3, 4, 5)
    .map { "Mambo no. \($0)" }
    .subscribe { print($0) }
```

```
next(Mambo no. 3)
next(Mambo no. 4)
next(Mambo no. 5)
completed
```

flatMap

```
func fetchUser(id: Int) -> Observable<User> {
    return URLSession.shared.rx // ...
}

Observable<Int>.of(1, 2, 3)
    .flatMap { fetchUser(id: $0) }
    .subscribe { print($0) }
```

```
next(User(id: 1))
next(User(id: 2))
next(User(id: 3))
completed
```

RxSwift Cheatsheet

TRANSFORMING OBSERVABLES

flatMapFirst

```
Observable<Int>.of(1, 2, 3)
    .flatMapFirst { fetchUser(id: $0) }
    .subscribe { print($0) }
```

```
next(User(id: 1))
completed
```

flatMapLatest

```
Observable<Int>.of(1, 2, 3)
    .flatMapLatest { fetchUser(id: $0) }
    .subscribe { print($0) }
```

```
next(User(id: 3))
completed
```

FILTERING OBSERVABLE

filter

```
Observable<Int>.of(1, 2, 3, 4, 5, 6)
    .filter { $0 > 4 }
    .subscribe { print($0) }
```

```
next(5)
next(6)
completed
```

distinctUntilChanged

```
Observable<Int>.of(1, 1, 1, 3, 3, 5, 1, 5)
    .distinctUntilChanged()
    .subscribe { print($0) }
```

```
next(1)
next(3)
next(5)
next(1)
next(5)
completed
```

skip(_: Int)

```
Observable<Int>.of(200, 300, 400, 500, 600)
    .skip(3)
    .subscribe { print($0) }
```

```
next(500)
next(600)
completed
```

CONDITIONAL AND BOOLEAN OPERATORS

takeWhile

```
Observable<Int>.of(1, 2, 3, 4, 5, 6)
    .takeWhile { $0 < 4 }
    .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

ERROR HANDLING OPERATORS

catchError

```
Observable<String>.error(RxError.unknown)
    .catchError({ error -> Observable<String> in
        return .just("RxSwift")
    })
    .subscribe { print($0) }
```

```
next(RxSwift)
completed
```

catchErrorJustReturn

```
Observable<String>.error(RxError.unknown)
    .catchErrorJustReturn("RxSwift")
    .subscribe { print($0) }
```

```
next(RxSwift)
completed
```


RxSwift Cheatsheet

COMBINING OBSERVABLES

merge

```
let letters = PublishSubject<String>()
let numbers = PublishSubject<String>()

Observable
    .merge(letters, numbers)
    .subscribe(onNext: { print($0) })

letters.onNext("A")
letters.onNext("B")
numbers.onNext("1")
numbers.onNext("2")
letters.onNext("AB")
numbers.onNext("3")
```

combineLatest

```
let letters = PublishSubject<String>()
let numbers = PublishSubject<String>()

Observable
    .combineLatest(letters, numbers) { "\($0) \($1)" }
    .subscribe(onNext: { print($0) })

letters.onNext("A")
letters.onNext("B")
numbers.onNext("1")
numbers.onNext("2")
letters.onNext("AB")
numbers.onNext("3")
```

zip

```
let letters = PublishSubject<String>()
let numbers = PublishSubject<String>()

Observable.zip(letters, numbers) { "\($0) \($1)" }
    .subscribe(onNext: { print($0) })

letters.onNext("A")
letters.onNext("B")
numbers.onNext("1")
numbers.onNext("2")
letters.onNext("AB")
numbers.onNext("3")
```

AGGREGATE OPERATORS

reduce

```
Observable<Int>.of(1, 2, 3, 4, 5)
    .reduce("") { $0.appending("\($1)") }
    .subscribe { print($0) }
```

```
next(12345)
completed
```

toArray

```
Observable.of(1, 2, 3, 4, 5)
    .toArray()
    .subscribe { print($0) }
```

```
next([1, 2, 3, 4, 5])
completed
```

LIFECYCLE

Cold observable

```
let coldObservable = Observable.just("Hello Rx!")
let subscription = coldObservable
    .subscribe { print($0) }
    .disposed(by: disposeBag)

// Subscription will be disposed after completed
// event.
```

Hot observable

```
let hotObservable = button.rx.tap.asObservable()
let subscription = hotObservable
    .subscribe { print($0) }
    .disposed(by: disposeBag)

// Subscription will be disposed when disposeBag
// will be deallocated.
```

Passion EL