RxSwift

A gentle introduction

Guille González

@gonzalezreal

http://reactivex.io

“RxSwift, A gentle introduction” - Guille González **@gonzalezreal**

Observer

on steroids

“RxSwift, A gentle introduction” - Guille González
@gonzalezreal

Taps, keyboard events, timers, GPS events, web service responses

UI update, data written to disk, API request, etc.

Asynchronous Events in Cocoa

→ Target-Action

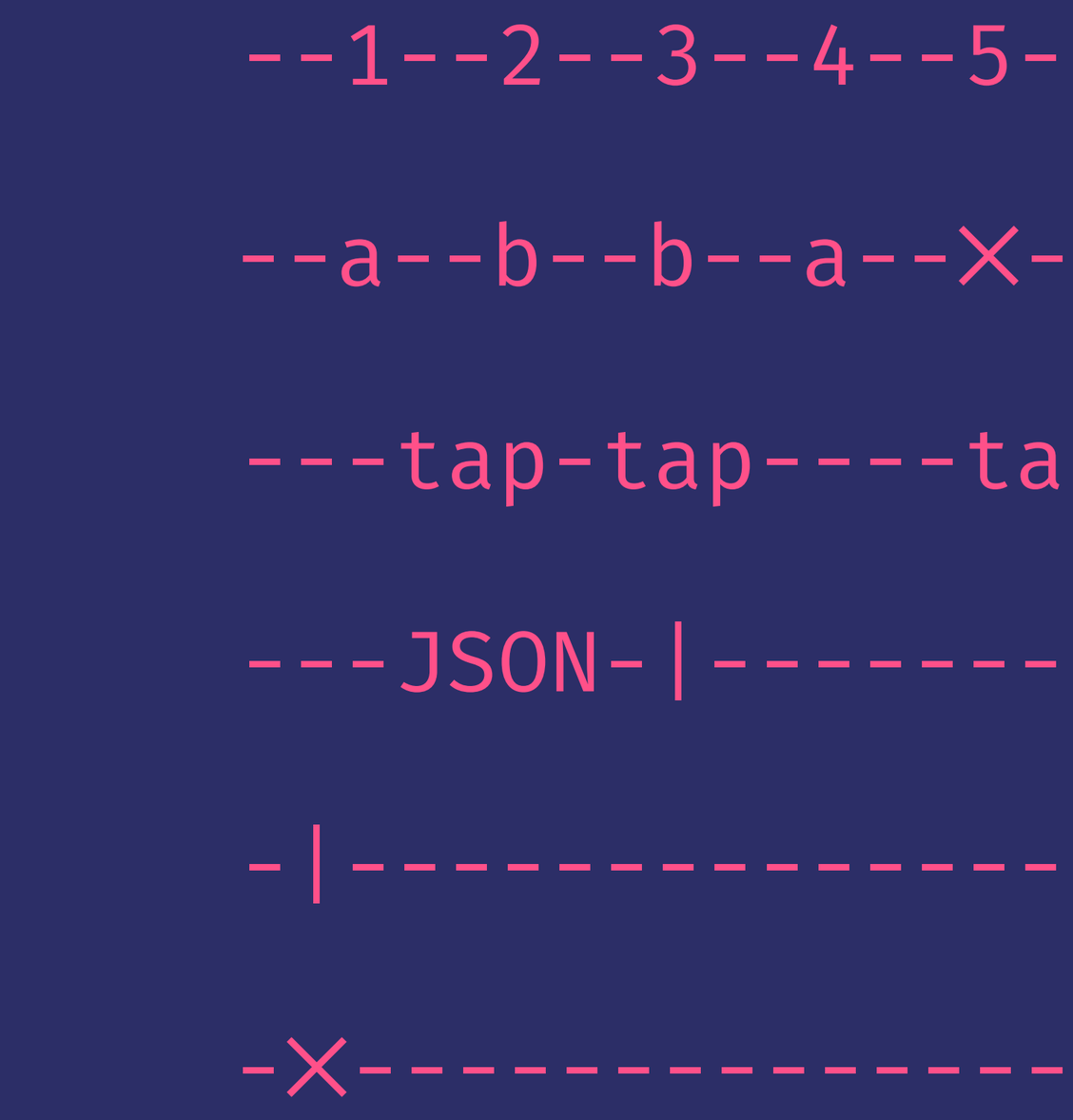
→ `NSNotificationCenter`

→ Key-Value Observing

→ Delegates

→ Callback Closures

Observable<Element>



“RxSwift, A gentle introduction” - Guille González **@gonzalezreal**

```
enum Event<Element>  {
    case Next(Element)
    case Error(ErrorType)
    case Completed
}
```

```
--Next("a")--Next("b")--Error(diskError)->

------Next---Next--------Next---------------->

---Next(json)-Completed--------------------->
```

Creating Observables

→ `Observable.empty()`

→ `Observable.just("🏀")`

→ `Observable.of("🏀", "🎾", "⚽")`

→ `Observable.error(Error.CouldNotDecodeJSON)`

Creating Observables

```
let o = Observable.create { observer in
    observer.on(.Next("👋 world!"))
    observer.on(.Completed)
    return NopDisposable.instance
}
```

If a tree falls in a forest and no one is around to hear it, does it make a sound?

— George Berkeley

Hot Observables	Cold observables
Use resources even when there are no subscribers.	Don't use resources until there is a subscriber.
Resources usually shared between all the subscribers.	Resources usually allocated per subscriber.
Usually stateful.	Usually stateless.
UI controls, taps, sensors, etc.	HTTP request, async operations, etc.

Observers

```
protocol ObservableType {
    func subscribe(on: (event: Event) -> Void) -> Disposable
}
```

Observers

```
Observable.create { observer in
    observer.onNext("👋 world!")
    observer.onCompleted()
    return NopDisposable.instance
}.subscribe { event in
    print(event)
}

// outputs:
//    Next(👋 world!)
//    Completed
```

Observers

```
Observable.create { observer in
    observer.onNext("👋 world!")
    observer.onCompleted()
    return NopDisposable.instance
}.subscribeNext { text in
    print(text)
}

// outputs:
//    👋 world!
```

Disposables

```
let appleWeb = Observable.create { observer in
    let task = session.dataTaskWithURL(appleURL) { data, response, error in
        if let data = data {
            observer.onNext(data)
            observer.onCompleted()
        } else {
            observer.onError(error ?? Error.UnknownError)
        }
    }

    task.resume()

    return AnonymousDisposable {
        task.cancel()
    }
}
```

Dispose Bags

```
self.disposeBag = DisposeBag()

...

appleWeb.subscribeNext { data in
    print(data)
}.addDisposableTo(disposeBag)
```

Operators

→ `map`

→ `flatMap`

→ `filter`

→ `throttle`

→ `merge`

→ `combineLatest`

→ and many more...

“RxSwift, A gentle introduction” - Guille González **@gonzalezreal**

map & flatMap

```
struct Country {
    let name: String
    let borders: [String]
}

protocol CountriesAPI {
    func countryWithName(name: String) -> Observable<Country>
    func countriesWithCodes(codes: [String]) -> Observable<[Country]>
}
```

map & flatMap

```
myAPI.countryWithName("spain")
     .flatMap { country in
         myAPI.countriesWithCodes(country.borders)
     }
     .map { countries in
         countries.map { $0.name }
     }
     .subscribeNext { countryNames in
         print(countryNames)
     }
```

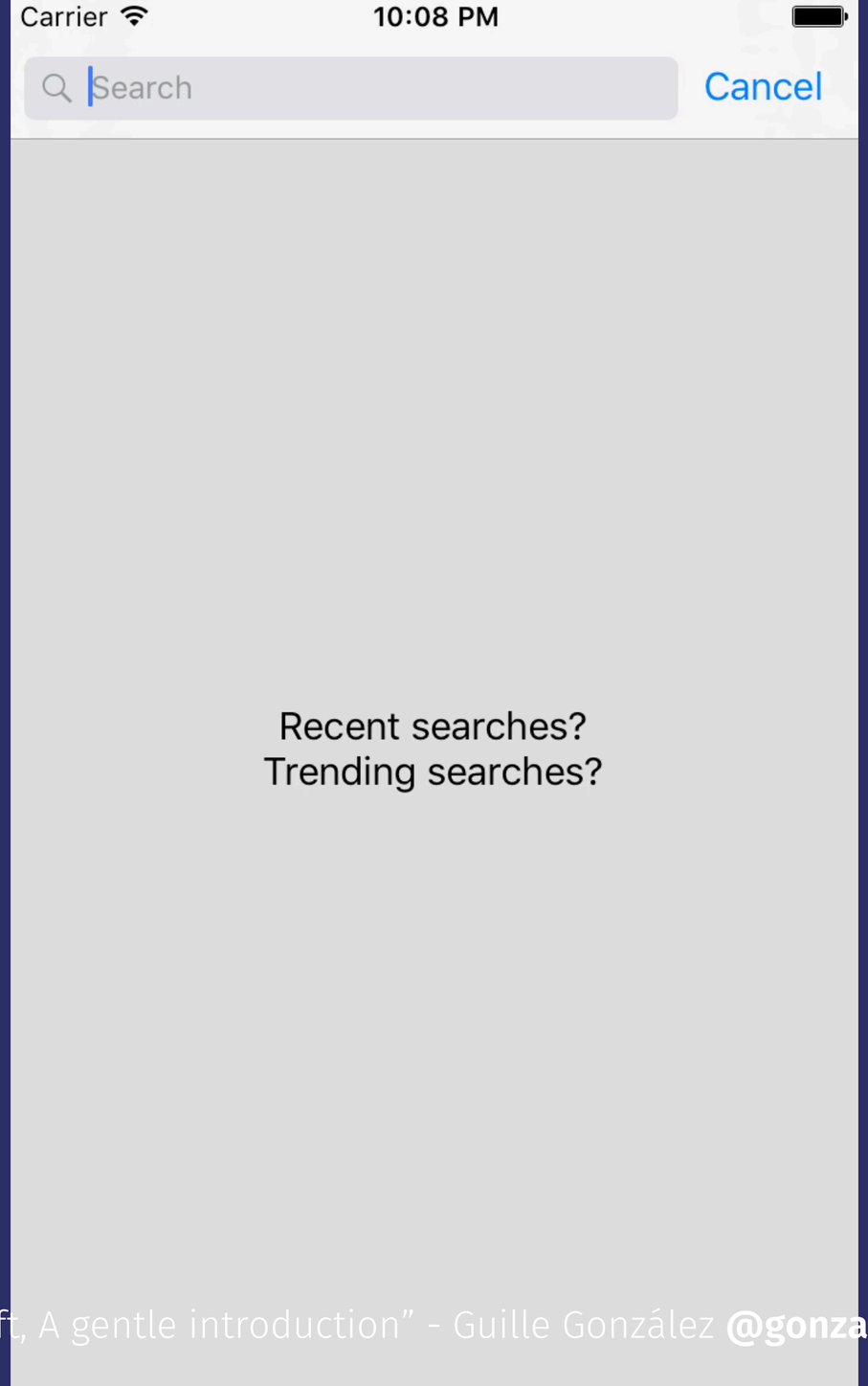
`Observable` chaining is similar to `Optional` chaining:

```
let cell = UITableViewCell(style: .Default, reuseIdentifier: nil)

let maybeSize  = cell.imageView?.image?.size
let maybeSize2 = cell.imageView.flatMap { $0.image }.flatMap { $0.size }
```

observeOn

```
myAPI.countryWithName("spain")
    .flatMap { country in
        myAPI.countriesWithCodes(country.borders)
    }
    .map { countries in
        countries.map { $0.name }
    }
    .observeOn(MainScheduler.instance)
    .subscribeNext { countryNames in
        // Main thread, all good
    }
```


Carrier
10:08 PM
Search
Cancel
Recent searches?
Trending searches?

throttle

```
let results = searchBar.rx_text
    .throttle(0.3, scheduler: MainScheduler.instance)
    .flatMapLatest { query in
        if query.isEmpty {
            return Observable.just([])
        }

        return searchShows(query)
    }
    .observeOn(MainScheduler.instance)
    .shareReplay(1)
```

combineLatest

```
Observable.combineLatest(emailField.rx_text, passwordField.rx_text)
{ email, password in
    return email.characters.count > 0 &&
    password.characters.count > 0
}
.bindTo(sendButton.rx_enabled)
.addDisposableTo(disposeBag)
```

Follow-up Resources

→ reactivex.io

→ rxmarbles.com

→ github.com/ReactiveX/RxSwift

→ tinyurl.com/consuming-web-services

Questions?
Comments?