Advanced RxSwift – Day 3

Younghwan Kim

RxSwift Basics

- Day 1 – Observable, Operator (Filter, Transform, Combine)
- Day 2 – Subject (flatMap, flatMapFirst, flatMapLatest)
- Day 3 – Two VCs communications with Subject, RxCocoa (Button)
- Day 4 – Sequential, Merged Observable Calls
- Day 5 – RxCocoa, UI Binding (Button, TextField, Label, TableView)

Advanced RxSwift

- Day 1 – Protocol-Oriented Programming, Protocol Extension, Associatetype
- Day 2 – Network Call, Generic Enum
- **Day 3 – Binding Track Activity (show / hide ‘Loading’), Scan Operator**

- Day 4 – Adding a Reactive Extension to Custom UI Element,
- 2 Way Binding, Advanced TableView – RxDataSources
- Day 5 – Schedulers (observeOn, subscribeOn),
Unit Test (RxTest, RxBlocking)

ActivityIndicator

https://github.com/ReactiveX/RxSwift/blob/master/RxExample/RxExample/Services/ActivityIndicator.swift

```
public class ActivityIndicator : SharedSequenceConvertibleType {

…

}
```

ActivityIndicator in ViewModel

```
class SimpleViewModel {
    // Is signing process in progress
    let signingIn: Observable<Bool>
    let signingInIndicator = ActivityIndicator()

    init() {
        self.signingIn = signingInIndicator.asObservable()
    }

    func simpleObservable() -> Observable<String> {
        return Observable<String>.create { observer in
            DispatchQueue.main.asyncAfter(deadline: .now() + 5) {
                observer.onNext("strawberry")
                observer.onCompleted()
            }
            return Disposables.create()
        }
    }
}
```

ActivityIndicator : UI Binding 1

```
class ViewController: UIViewController {
    @IBOutlet weak var TrackActivityOulet: UIActivityIndicatorView!
    @IBOutlet weak var backgroundView: UIImageView!
    @IBOutlet weak var trackActivityButton: UIButton!
    var disposeBag = DisposeBag()
    let viewModel = SimpleViewModel()

    override func viewDidLoad() {
        super.viewDidLoad()


        viewModel.signingIn
            .bind(to: TrackActivityOulet.rx.isAnimating)
            .disposed(by: disposeBag)

        viewModel.signingIn
            .map { !$0 }
            .bind(to: backgroundView.rx.isHidden)
            .disposed(by: disposeBag)
```

ActivityIndicator : UI Binding 2

```
    viewModel.signingIn
        .bind(to: trackActivityButton.rx.isHidden)
        .disposed(by: disposeBag)

    trackActivityButton.rx.tap.asDriver()
        .drive(onNext: { [weak self] _ in
            self?.activityButtonAction()
        }).disposed(by: disposeBag)
}

func activityButtonAction() {
    self.viewModel.simpleObservable()
        .observeOn(MainScheduler.instance)
        .trackActivity(viewModel.signingInIndicator)
        .subscribe(onNext: { _ in
        })
        .disposed(by: self.disposeBag)
}
```

Scan

The Scan operator applies a function to the first item emitted by the source Observable and then emits the result of that function as its own first emission.

It also feeds the result of the function back into the function along with the second item emitted by the source Observable in order to generate its second emission.

It continues to feed back its own subsequent emissions along with the subsequent emissions from the source Observable in order to create the rest of its sequence.

This sort of operator is sometimes called an “accumulator” in other contexts.

Scan

scan takes two parameters:

- initial value - you can think of it as the first value of your state
- closure(lastState, newValue) - scan runs that closure each time it gets a new value - it calls it with two parameters: the last state you had and the value that was just emitted.

Scan - Creating a boolean switch

```
//(tap) -> Void -> (scan) -> Bool -> (subscribe)
        self.switchButton.rx.tap
            .scan(false) { lastState, newValue in
                return !lastState
            }
            .subscribe(onNext: { value in
                print("tap: \(value)")
            }).disposed(by: disposeBag)
```

Scan - Creating a counter

```
//(tap) -> Void -> (scan) -> Int -> (subscribe)
        self.counterButton.rx.tap
            .scan(0) { lastCount, newValue in
                return lastCount + 1
            }
            .subscribe(onNext: { value in
                print("taps: \(value)")
            }).disposed(by: disposeBag)
```

Scan - Getting the last N values

```
let numbers = Observable.from([0, 1, 2 , 3, 4, 5, 6])

        //(numbers) -> Int -> (scan) -> [Int] -> subscribe
        numbers.scan([]) { lastSlice, newValue in
                return Array(lastSlice + [newValue]).suffix(3)
            }
            .subscribe(onNext: { value in
                print("last 3: \(value)")
            }).disposed(by: disposeBag)
```

Lab

Add ActivityIndicator in Borders App in day 2