Subjects

Outline

1. What are the subjects?
2. PublishSubject
3. BehaviorSubject
4. ReplaySubject
5. BehaviorRelay

1. What are subjects?

- Subjects act as both an observable and an observer.

- The subject received `.next` events, and every time it receives an event, it turns around and emits it to its subscribers.

1. What are subjects?

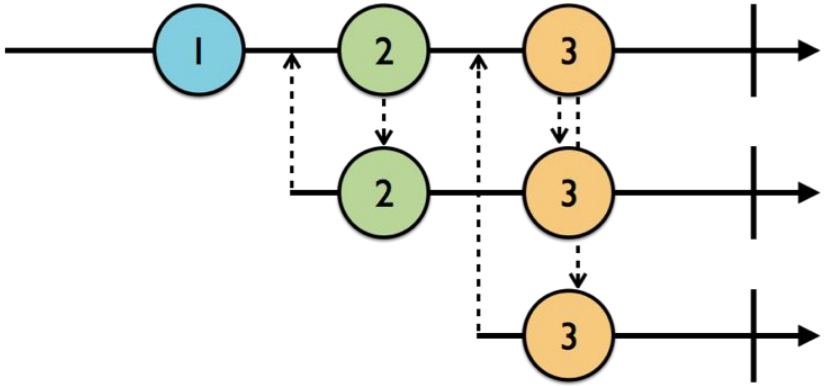
- There are four subjects types in RxSwift, and two relay types that wrap subjects:

  - PublishSubject
  - BehaviorSubject
  - ReplaySubject
  - AsyncSubject
  - PublishRelay and BehaviorRelay

2. PublishSubject

- PublishSubject starts empty and only emits new elements to subscribers.

- PublishSubject comes handy when you simply want subscribers to be notified of new events from the point at which they subscribed, until they either unsubscribe, or the subject has terminated with a `.completed` or `.error` event.



2. PublishSubject (cont)

- Example:

```
let bag = DisposeBag()
let publishSubject = PublishSubject<String>()

publishSubject.onNext("Emit 1")

let subscriberOne = publishSubject.subscribe { element in
    print("subscriber 1: \(element)")
}
subscriberOne.disposed(by: bag)

publishSubject.onNext("Emit 2")
publishSubject.onNext("Emit 3")

/* Prints:
subscriber 1: next(Emit 2)
subscriber 1: next(Emit 3)
*/
```

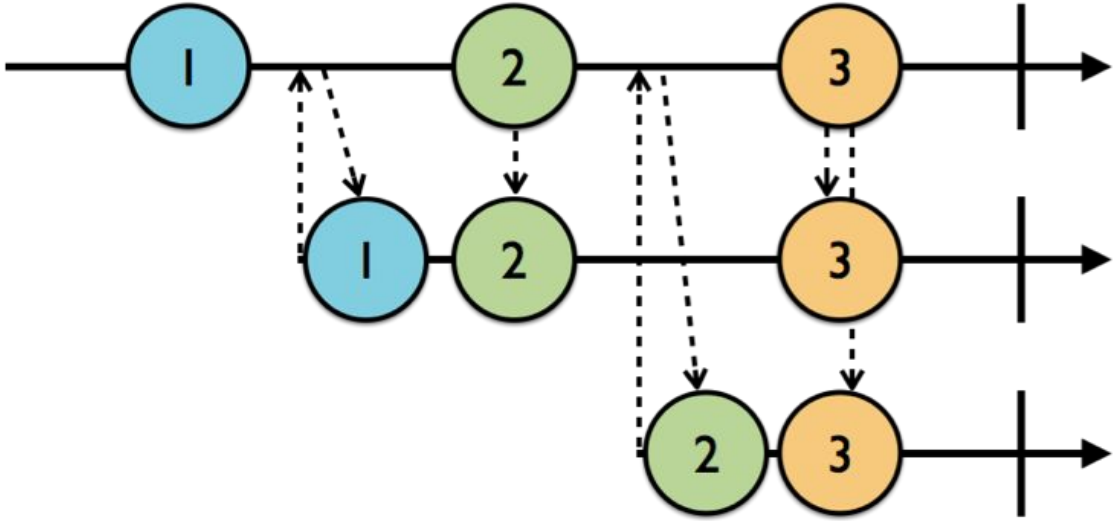
2. PublishSubject (cont)

- **Note**: All terminated subjects can still re-emit stop event to new subscribers.

```
let bag = DisposeBag()
let publishSubject = PublishSubject<String>()

publishSubject.onNext("Emit 1")

publishSubject
    .subscribe { element in
        print("subscriber 1: \(element)")
    }
    .disposed(by: bag)

publishSubject.onNext("Emit 2")
publishSubject.onNext("Emit 3")
publishSubject.onCompleted()

print("- - - - - - -")

publishSubject
    .subscribe { element in
        print("subscriber 2: \(element)")
    }
    .disposed(by: bag)
```

3. BehaviorSubject:

- **BehaviorSubject** works as same as **PublishSubject** except it has initial value and replay that value or .next the last event of observable to a new subscriber.



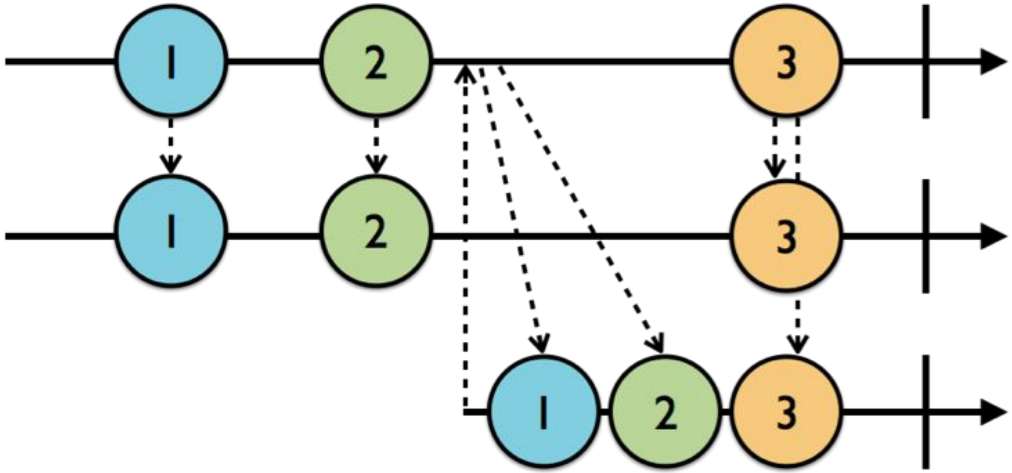
3. BehaviorSubject (cont)

- Example:

```
let bag = DisposeBag()
let behaviorSubject = BehaviorSubject<String>(value: "Initial Value")

behaviorSubject.onNext("Emit 1")

print("- Subscribe here -")
behaviorSubject
    .subscribe { element in
        print("Subscriber: \(element)")
    }
    .disposed(by: bag)

behaviorSubject.onNext("Emit 2")
behaviorSubject.onNext("Emit 3")

/*Prints
- Subscribe here -
Subscriber: next(Emit 1)
Subscriber: next(Emit 2)
Subscriber: next(Emit 3)
*/
```

4. ReplaySubject:

- **ReplaySubject** initializes with a buffer size that is the number of most recent emissions.
- It will replay all emissions in the buffer to the subscribers when they subscribe.



4. ReplaySubject (cont)

- Example:

```
let bag = DisposeBag()
let replaySubject = ReplaySubject<String>.create(bufferSize: 2)
// Initialize a ReplaySubject with buffer size = 2
// If you want to replay all emissions, consider to use createUnbounded()

replaySubject.onNext("Emit 1")
replaySubject.onNext("Emit 2")
replaySubject.onNext("Emit 3")

print("- Before subscribe -")
replaySubject
    .subscribe { element in
        print("Subscriber: \(element)")
    }
    .disposed(by: bag)
print("- After subscribe -")

replaySubject.onNext("Emit 4")
replaySubject.onNext("Emit 5")

/*Prints
- Before subscribe -
Subscriber: next(Emit 2)
Subscriber: next(Emit 3)
- After subscribe -
Subscriber: next(Emit 4)
Subscriber: next(Emit 5)
*/
```

5. BehaviorRelay:

- **BehaviorRelay** is a wrapper class for **BehaviorSubject**.

- It save current data of **BehaviorSubject** as a state and replay only the initialized or the latest value to the subscribers.

5. BehaviorRelay (cont)

- Example:

```
let bag = DisposeBag()
let behaviorRelay = BehaviorRelay<Bool>(value: false)

behaviorRelay
    .asObservable()
    .subscribe { element in
        print("Subscriber: \(element)")
    }
    .disposed(by: bag)

behaviorRelay.accept(false)
behaviorRelay.accept(true)

/*Prints:
Subscriber: next(false)
Subscriber: next(false)
Subscriber: next(true)
*/
```

Question & Answer?