RXSwift - Subjects

Ref link:
https://www.raywenderlich.com/books/rxswift-reactive-programming-with-swift/v4.0/chapters/3-subjects
https://fxstudio.dev/rxswift-hello-subjects/

✓ Definition:

Observables are a fundamental part of RxSwift, but they're essentially read-only. You may only subscribe to them to get notified of new events they produce.

A common need when developing apps is to manually add new values onto an observable during runtime to emit to subscribers. That's why we need **Subject**

- Subject can act as both **Observable sequence & Observer**
  - An **Observable sequence**, which means it can be subscribed to
  - An **Observer** that enables adding new elements onto a subject that will then be emitted to the subject subscribers

✓ Type of subjects:

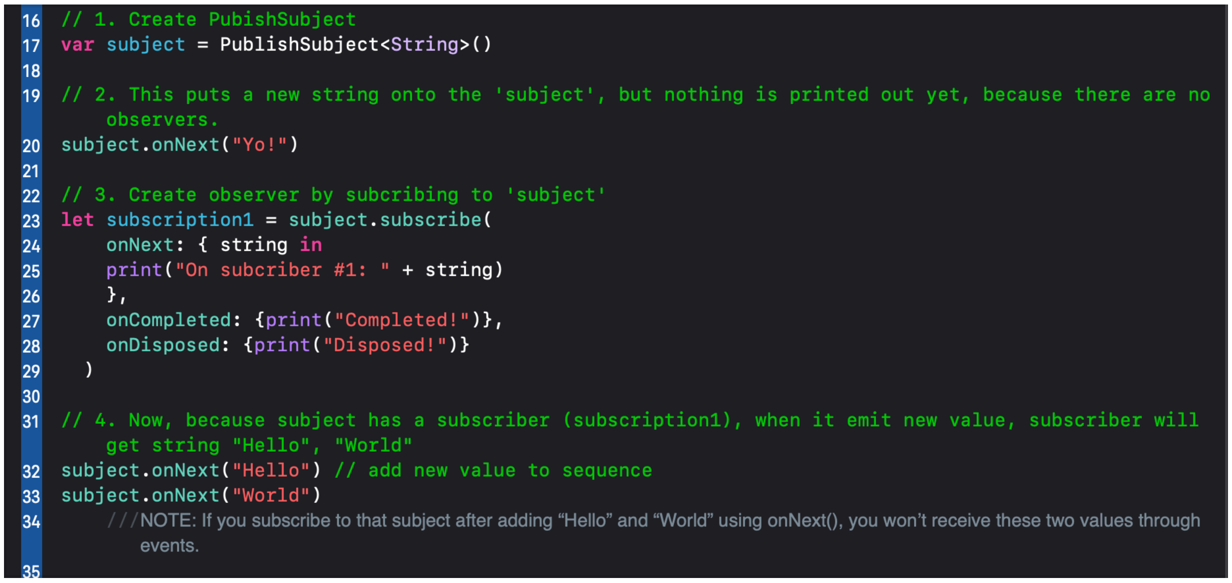
PublishSubject	Starts empty and only emits new elements to subscribers
BehaviorSubject	Starts with an initial value and replays it or the latest element to new subscribers.
ReplaySubject	Initialized with a buffer size and will maintain a buffer of elements up to that size and replay it to new subscribers.

AsyncSubject	Emits only the last ***next*** event in the sequence, and only when the subject receives a completed event. This is a seldom used kind of subject
Variable	
PublishRelay & BehaviorRelay	These wrap their respective subjects, but only accept and relay next events.
	You cannot add a completed or error event onto relays at all, so they're great for non-terminating sequences.

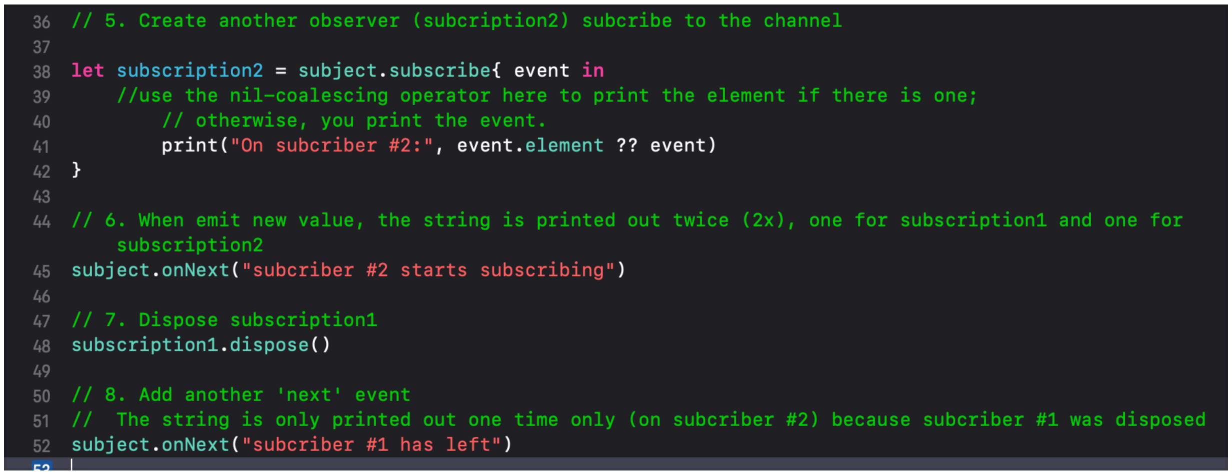
NOTE:
Emitting previous next events to new subscribers is called ***replaying****,*
and publish subjects ***DO NOT replay****.*

✓ **Publish Subject: https://fxstudio.dev/rxswift-publish-subjects/**

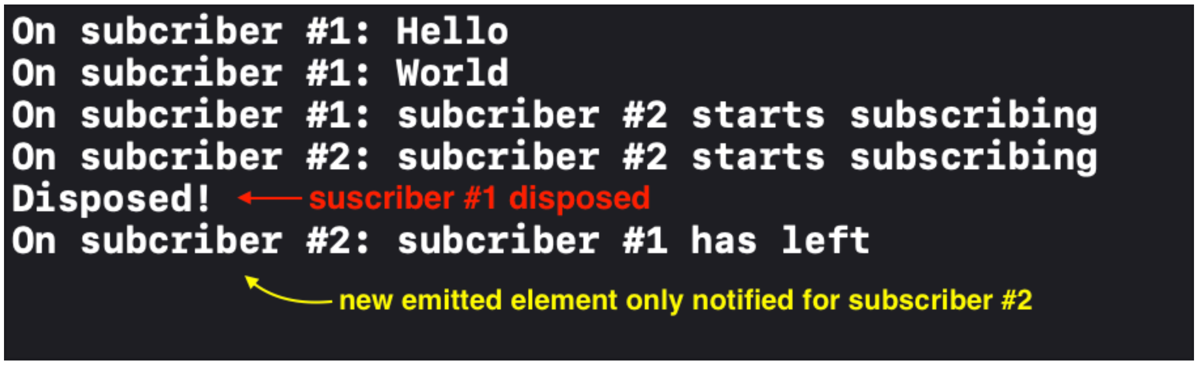
- **PublishSubject** will receive information and then publish it to subscribers.
  - It's of type String, so it can ONLY receive and publish strings.
  - After being initialized, it's ready to receive strings.
- Emits ONLY new next events to its subscribers.
  - Elements added to a **PublishSubject** before a subscriber subscribes will not be received by that subscriber



```swift
// 1. Create PublishSubject
var subject = PublishSubject<String>()

// 2. This puts a new string onto the 'subject', but nothing is printed out yet, because there are no observers.
subject.onNext("Yo!")

// 3. Create observer by subcribing to 'subject'
let subscription1 = subject.subscribe(
    onNext: { string in
    print("On subcriber #1: " + string)
    },
    onCompleted: {print("Completed!")},
    onDisposed: {print("Disposed!")}
  )

// 4. Now, because subject has a subscriber (subscription1), when it emit new value, subscriber will get string "Hello", "World"
subject.onNext("Hello") // add new value to sequence
subject.onNext("World")
    ///NOTE: If you subscribe to that subject after adding "Hello" and "World" using onNext(), you won't receive these two values through events.
```



```swift
// 5. Create another observer (subcription2) subcribe to the channel

let subscription2 = subject.subscribe{ event in
    //use the nil-coalescing operator here to print the element if there is one;
        // otherwise, you print the event.
        print("On subcriber #2:", event.element ?? event)
}

// 6. When emit new value, the string is printed out twice (2x), one for subscription1 and one for subscription2
subject.onNext("subcriber #2 starts subscribing")

// 7. Dispose subscription1
subscription1.dispose()

// 8. Add another 'next' event
//  The string is only printed out one time only (on subcriber #2) because subcriber #1 was disposed
subject.onNext("subcriber #1 has left")
```

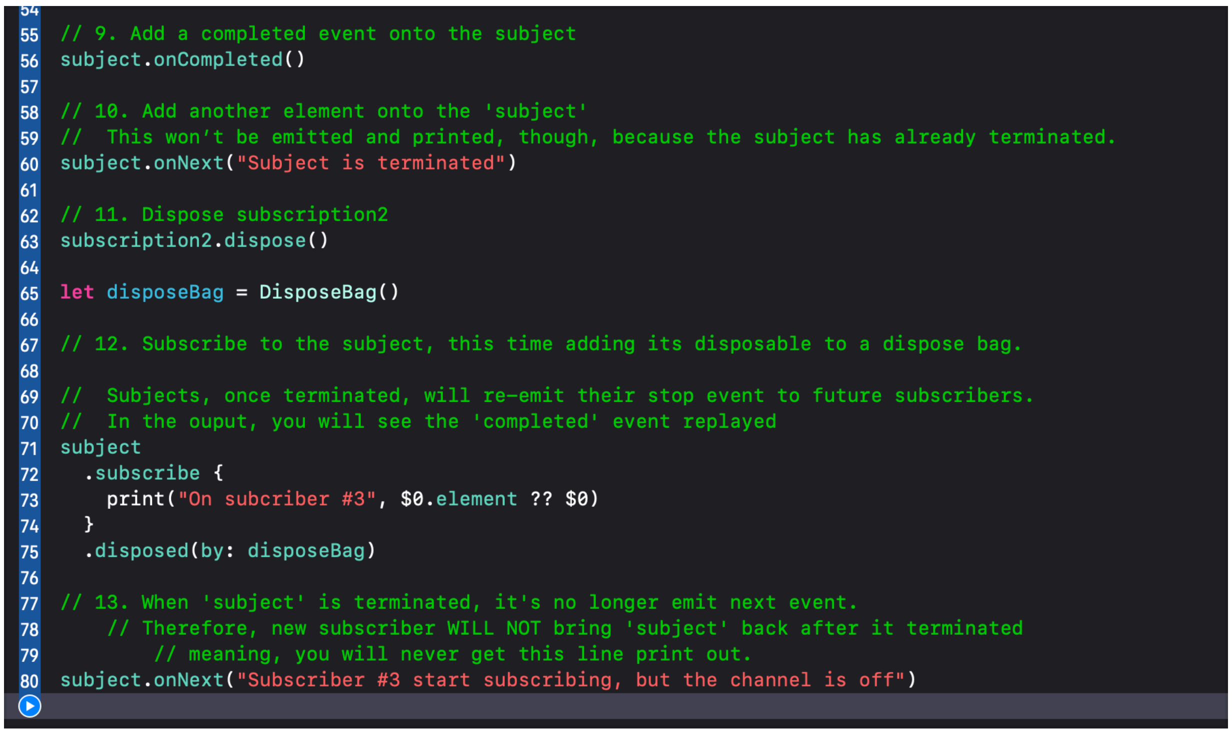


```
On subcriber #1: Hello
On subcriber #1: World
On subcriber #1: subcriber #2 starts subscribing
On subcriber #2: subcriber #2 starts subscribing
Disposed!
On subcriber #2: subcriber #1 has left
```

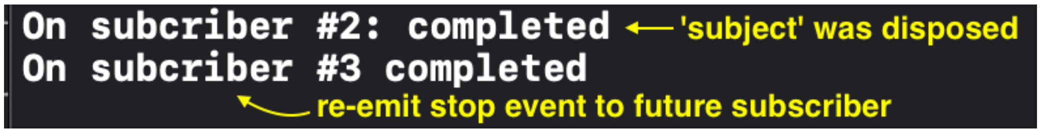
****NOTE**: Subscribers will be notified of new events from the point at which they subscribed, until either they unsubscribe, or the subject has terminated with a completed or error event.

- When a publish subject receives a **completed** or **error** event, also known as **a stop event**, it will emit that stop event to new subscribers and it will no longer emit next events.

EX: (continue the code from above)



```
// 9. Add a completed event onto the subject
subject.onCompleted()

// 10. Add another element onto the 'subject'
//  This won't be emitted and printed, though, because the subject has already terminated.
subject.onNext("Subject is terminated")

// 11. Dispose subscription2
subscription2.dispose()

let disposeBag = DisposeBag()

// 12. Subscribe to the subject, this time adding its disposable to a dispose bag.

//  Subjects, once terminated, will re-emit their stop event to future subscribers.
//  In the ouput, you will see the 'completed' event replayed
subject
  .subscribe {
    print("On subcriber #3", $0.element ?? $0)
  }
  .disposed(by: disposeBag)

// 13. When 'subject' is terminated, it's no longer emit next event.
    // Therefore, new subscriber WILL NOT bring 'subject' back after it terminated
        // meaning, you will never get this line print out.
subject.onNext("Subscriber #3 start subscribing, but the channel is off")
```



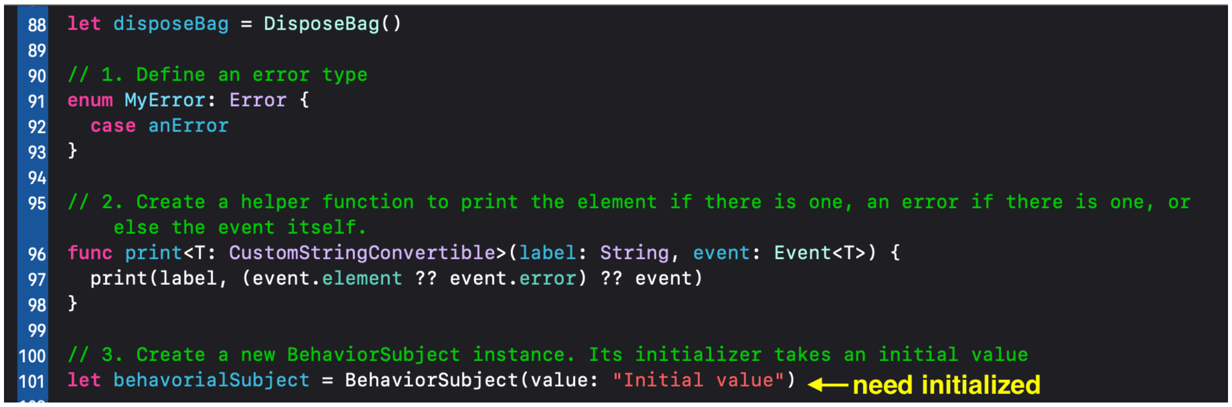
```
On subcriber #2: completed
On subcriber #3 completed
```

Behavior Subject: https://fxstudio.dev/rxswift-behavior-subjects/

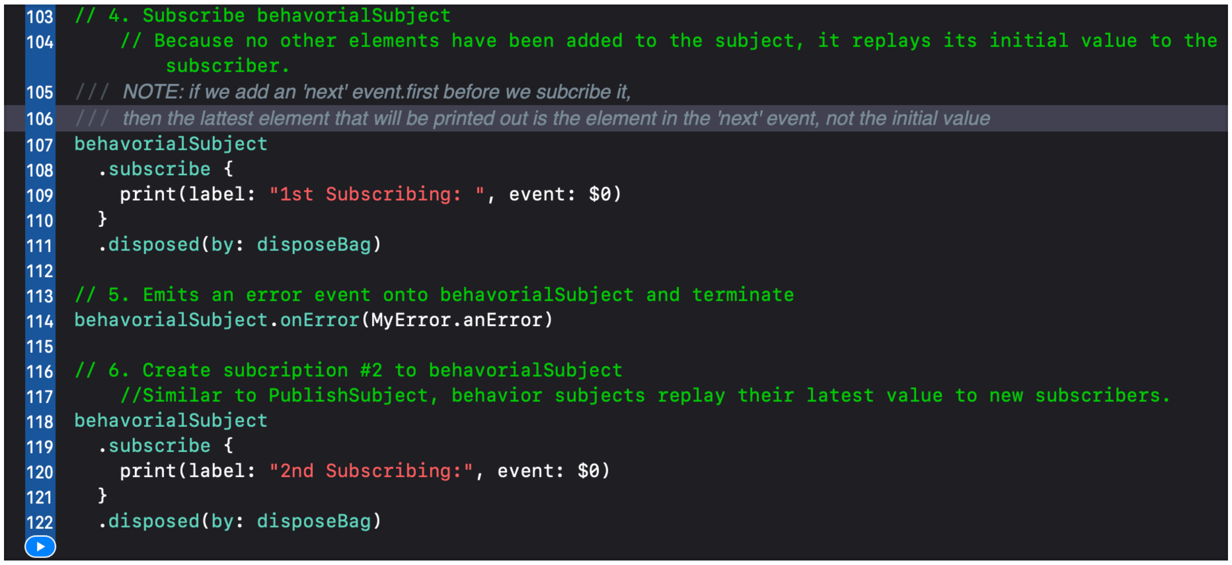
- **Behavior subjects** work similarly to publish subjects, except they will replay the latest next event to new subscribers
  - Subscribers will always **receive the most recent 'next' event** in the sequence even if they *subscribed after* that event was emitted

- A **BehaviorSubject** is *initialized* with a starting value
  - Because BehaviorSubject always emits its latest element, you can't create one without providing an initial value
    - If you can't provide an initial value at creation time, that probably means you need to use a **PublishSubject** instead, or model your element as an **Optional**.

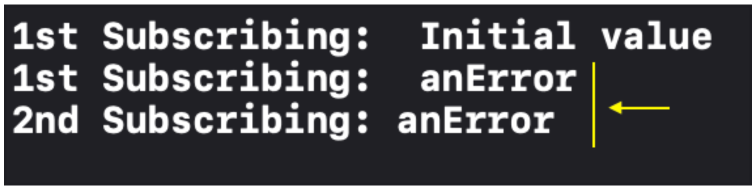
- Then, it **replays** to the new **subscribers** a 'next' event containing the most recent elements
- OR the initial value if no new recent elements have been added to it beforehand.



```swift
let disposeBag = DisposeBag()

// 1. Define an error type
enum MyError: Error {
  case anError
}

// 2. Create a helper function to print the element if there is one, an error if there is one, or
    else the event itself.
func print<T: CustomStringConvertible>(label: String, event: Event<T>) {
  print(label, (event.element ?? event.error) ?? event)
}

// 3. Create a new BehaviorSubject instance. Its initializer takes an initial value
let behavorialSubject = BehaviorSubject(value: "Initial value")
```



```swift
// 4. Subscribe behavorialSubject
    // Because no other elements have been added to the subject, it replays its initial value to the
       subscriber.
/// NOTE: if we add an 'next' event.first before we subcribe it,
/// then the lattest element that will be printed out is the element in the 'next' event, not the initial value
behavorialSubject
  .subscribe {
    print(label: "1st Subscribing: ", event: $0)
  }
  .disposed(by: disposeBag)

// 5. Emits an error event onto behavorialSubject and terminate
behavorialSubject.onError(MyError.anError)

// 6. Create subcription #2 to behavorialSubject
    //Similar to PublishSubject, behavior subjects replay their latest value to new subscribers.
behavorialSubject
  .subscribe {
    print(label: "2nd Subscribing:", event: $0)
  }
  .disposed(by: disposeBag)
```



```
1st Subscribing:  Initial value
1st Subscribing:  anError
2nd Subscribing: anError
```

- Usage:
  - Behavior subjects are useful when you want to pre-populate a view with the most recent data.
    - EX1: you could bind controls in a user profile screen to a behavior subject, so that the latest values can be used to pre-populate the display while the app fetches fresh data.
    - EX2: In a chat app, you might use a **BehaviorSubject** to

pre-fill a new posts title text field beginning with the initial name untitled.

✓ **Replay Subject: https://fxstudio.dev/rxswift-replay-subjects/**

- ReplaySubject - replay more than the most recent element on a sequence to new subscribers
- A **ReplaySubject** is ***initialized* with a buffer size** and that value cannot be changed after initialization.
- It will maintain a buffer up to the buffer size of the most recent next events,
  - It will replay the buffer to the new **subscribers** as if those events had happened immediately after each other
- It will also reemit its stop event to new subscribers
- EX:

You can use replay subject to display as many as the ***five most recent search items*** *whenever a search controller is presented.*

✓ **Variable**: https://fxstudio.dev/rxswift-relays/

- Variable is essentially **a wrapper** around **BehaviorSubject**
- A **variable** is guaranteed to never emit an error event and terminate. It also automatically completes when its about to be deallocated
- A variable uses the dot "." syntax to get the latest value or to set a new value onto it.
  - You can access a variable's **BehaviorSubject** by calling .**Observable()**