Subject

Subject클래스는 **Observable**과 **Observer**의 속성을 모두 가지고 있는 클래스이다.

실제로 RxJava에서 Subject는 Observable과 Observer 모두를 구현하고 있다.

```
public abstract class Subject<T> extends Observable<T> implemen
    ...
}
```

Observable처럼 데이터를 **subscribe** 할 수 있으며

Observer처럼 데이터를 onNext같은 메서드를 통해 **emit** 할 수도 있다는 의미다.

또한, **Cold Observable을 Hot Observable로 바꿔주는 역할**도 하고 있다.

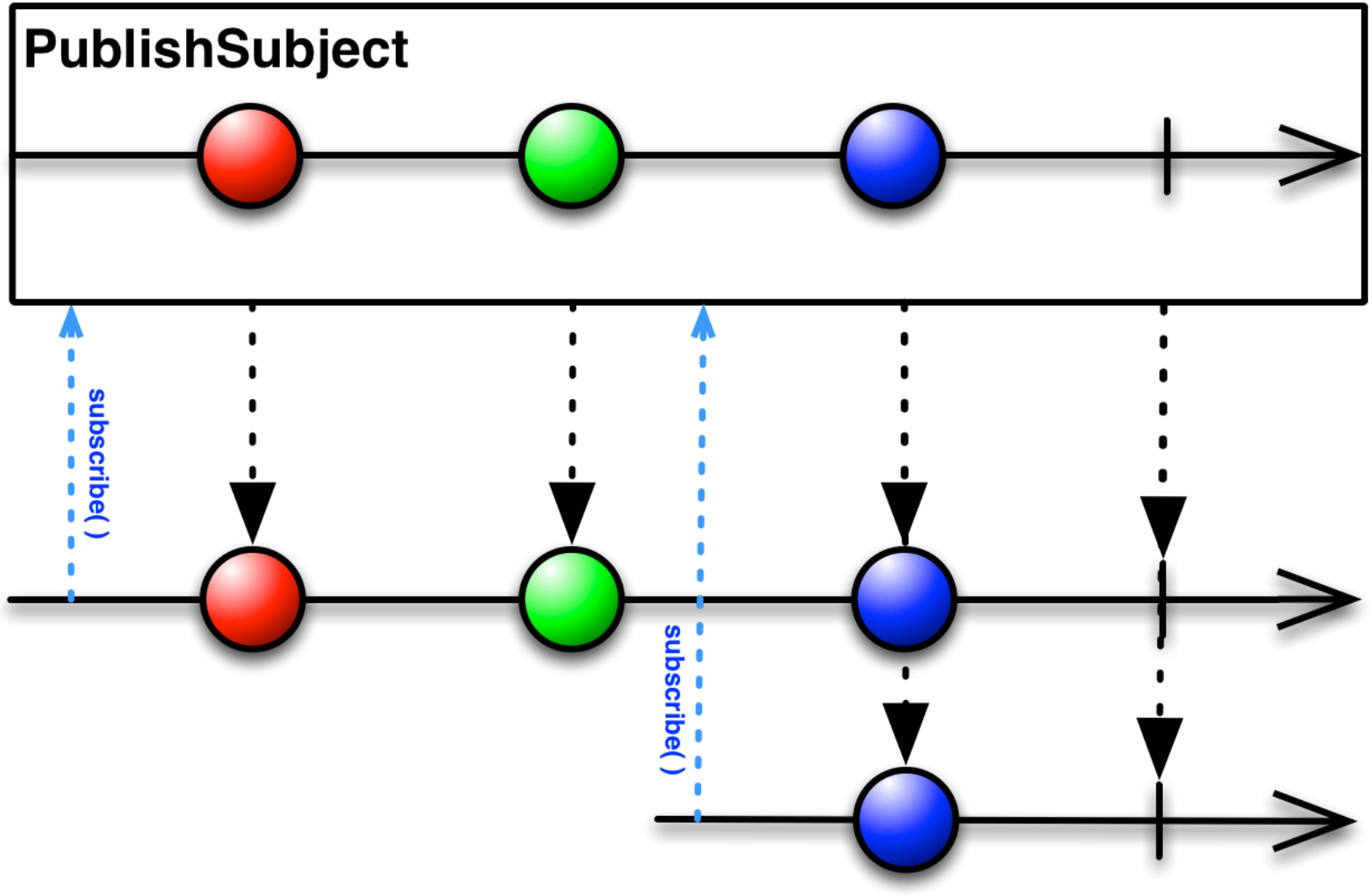
Subject는 4가지의 종류가 있다.

- PublishSubject
- BehaviorSubject
- AsyncSubject
- ReplaySubject

하나하나 알아보자

PublushSubject

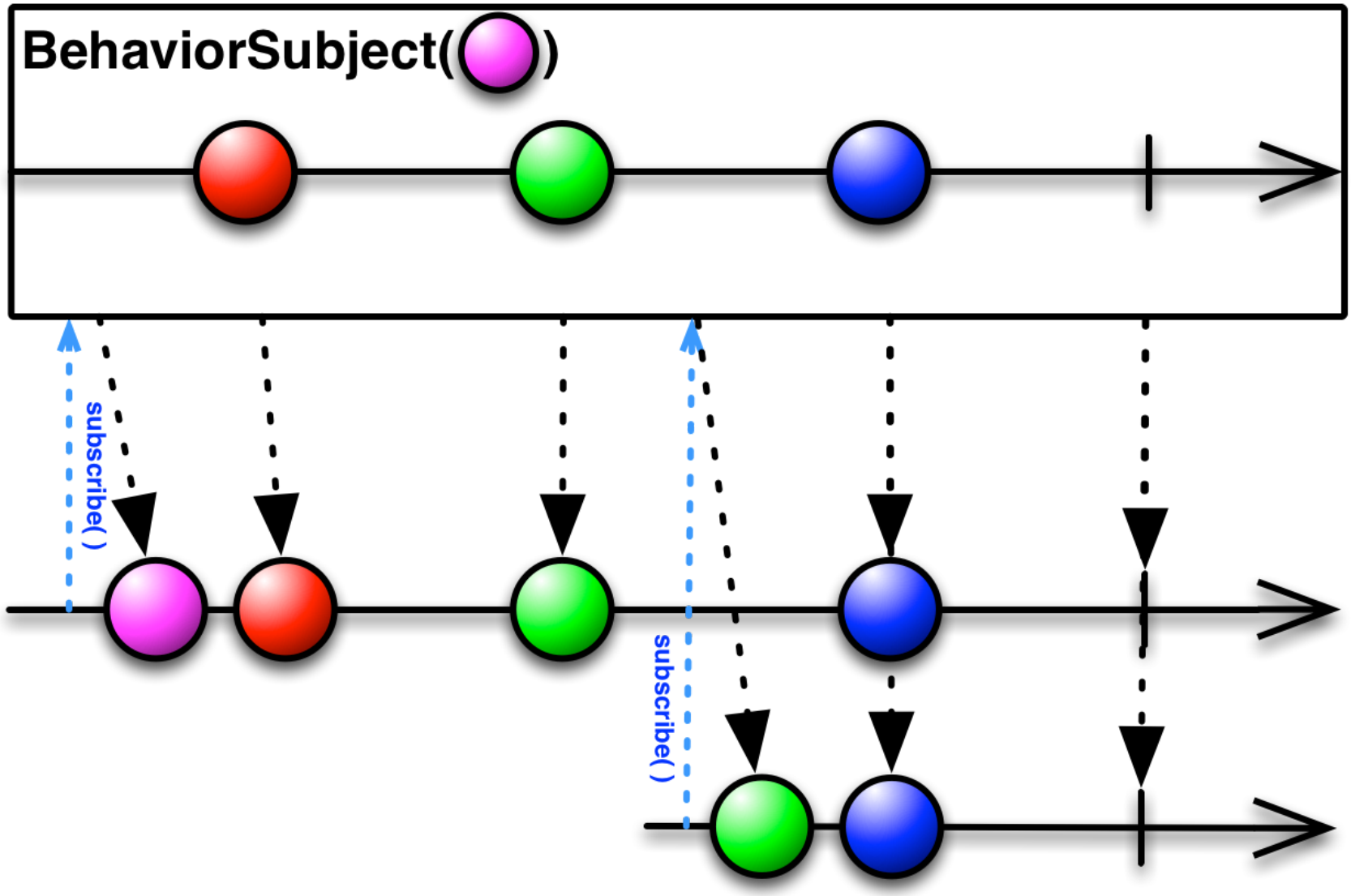
가장 일반적인 Subject클래스로 구독자가 **subscribe() 를 호출한 이후 시점**부터의 데이터를 전달받을 수 있다.



위 마블에서와 같이 첫 번째 타임라인에서는 모든 아이템을 전달받고, 두 번째 타임라인에서는 subscribe 이후인 파란공만 전달 받는다.

BehaviorSubject

BehaviorSubject클래스는 **subscribe시 가장 최근 값 혹은 기본값**을 넘겨주는 클래스이다. 여러개의 값들이 emit되었어도 오직 **가장 마지막에 emit된 값**을 넘겨준다. 기본값을 가지려면 createDefault 함수로 생성하면 된다.



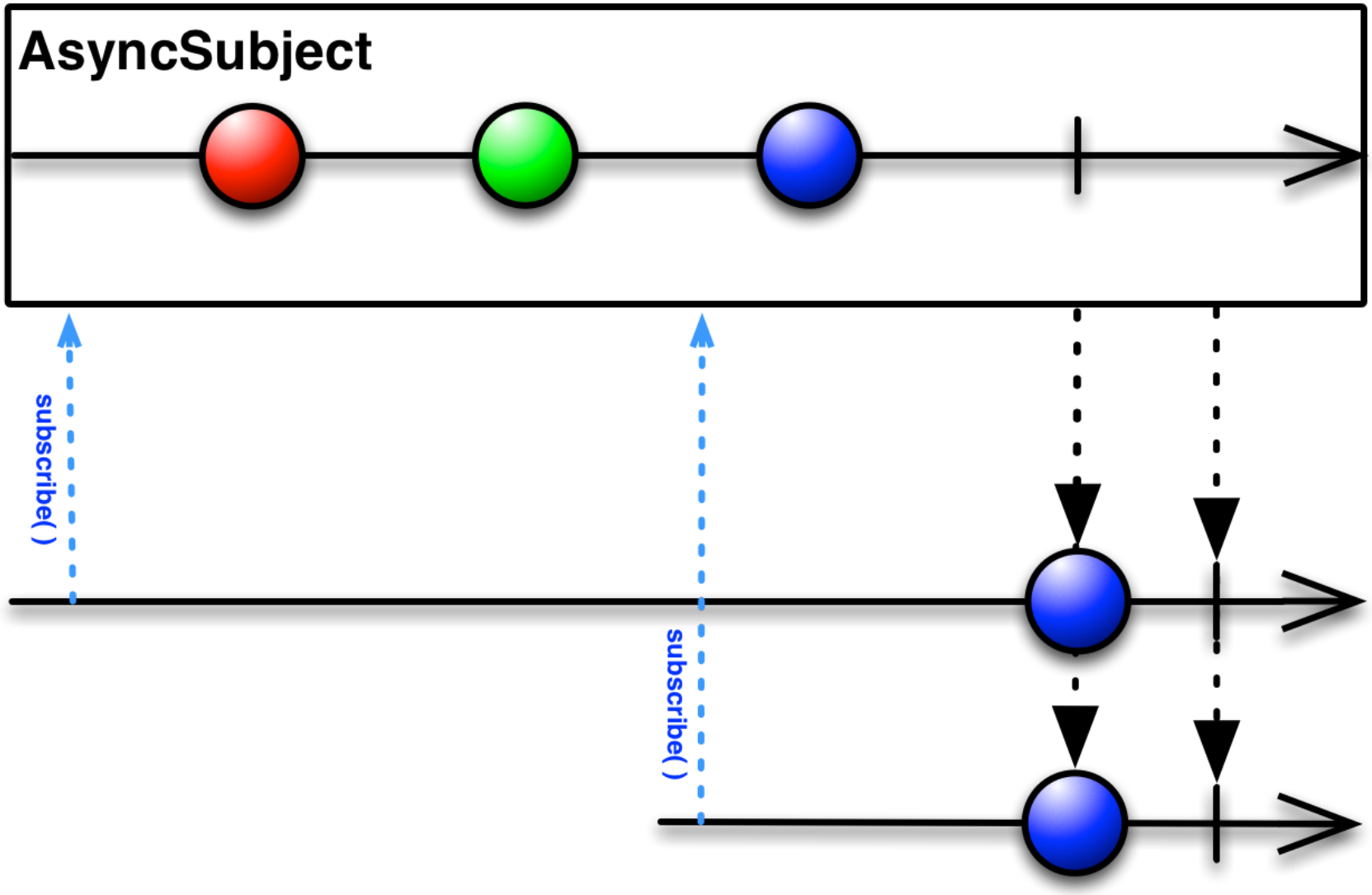
위 마블에서 첫 번째 타임라인에서 subcirbe시 초기값인 보라공을 전달 받고, 두 번째 타임라인에선 가장 최근에 emit된 값인 초록공을 전달받는다.

AsyncSubject

AsyncSubject클래스는 Observable에서 발행한 마지막 데이터를 얻어올 수 있는 Subject 클래스이다.

마지막으로 방출된 데이터를 기억하고 있다가 **onComplete()** 를 호출하면 그 값을 구독자에게 보낸다.

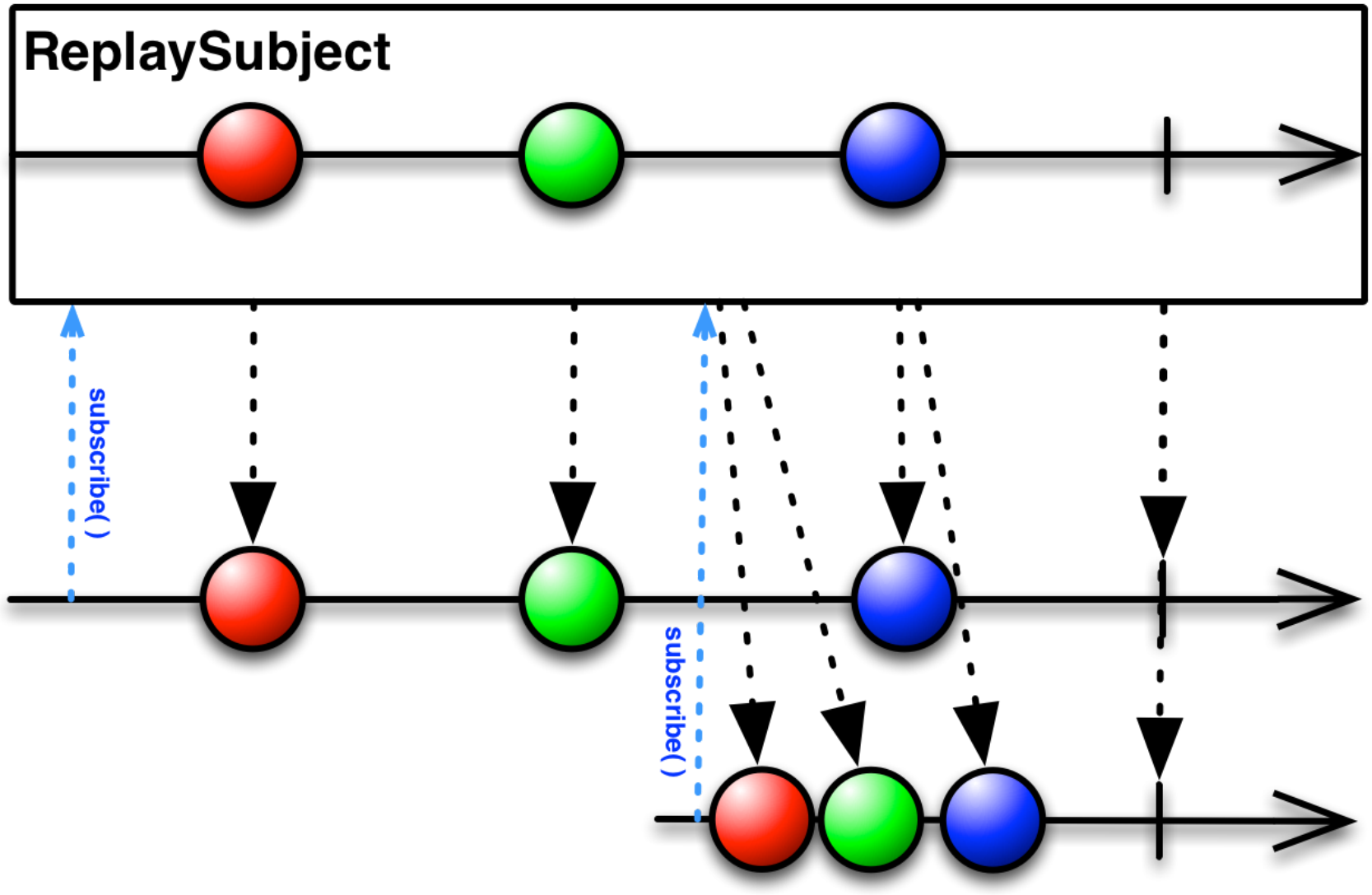
마지막 데이터가 아닌 데이터들은 무시당한다.



위 마블과 같이 완료되기 전까지는 구독자에게 데이터를 전달하지 않다가 완료됨과 동시에 모든 구독자에게 마지막 데이터를 발행하고 종료된다.

ReplaySubject

ReplaySubject클래스는 subscirbe시 subscribe를 시작한 시점과 관계 없이 subject가 **emit했던 모든 값들**을 전달받습니다. 이러한 특성 때문에 메모리 누수를 염두해두고 사용해야합니다.



위 마블에서 두 번째 타임라인과 같이 subscribe한 시점에서 그간 emit되었던 모든 데이터를 전달받습니다.

Relay

Subject는 위에서 살펴봤듯이 비 Rx적인 API들을 Rx답게 사용해 줄 수 있는 유용한 기능이다. 그러나 **Subject는 onComplete 또는 onError를 호출한 다음에는 더 이상 데이터를 emit 할 수 없다는 단점**을 가지고 있다.

Relay는 이런 Subject의 onComplete혹은 onError를 호출 한 후에는 더 이상 데이터를 emit 할 수 없다는 문제를 해결하기 위해 만든 라이브러리이다.

즉, Relay는 기본적으로 **Subject와 비슷한 특성**을 가지지만 **onComplete 및 onError가 없기 때문에 끊임없이 데이터를 emit 할 수 있다.**

Subject가 Observable과 Observer를 구현했던 것은 위에서 설명했었는데

Relay는 Observable과 **Consumer**을 구현하고있는 클래스이다.

```
public abstract class Relay<T> extends Observable<T> implements
    ...
}
```

Observer의 경우는

```
public interface Observer<T> {
    void onSubscribe(@NonNull Disposable d);

    void onNext(@NonNull T t);

    void onComplete();
}
```

위와 같이 onComplete를 가지고 있는 반면

Cunsumer의 경우는

```
public interface Consumer<T> {
    void accept(T t) throws Exception;
}
```

위와 같이 오로지 값을 전달 하는 accept만 가지고있다.

Relay의 종류는 Subject와 비슷하므로 다음과 같다

- PublishRelay
- BehaviorRelay
- ReplayRelay

각 Relay 모두 해당하는 이름의 Subject와 유사한 동작을 수행한다.

또한 Relay 자체를 구현해서 사용자가 원하는 Custom한 Relay구현도 가능하다.

다시 한번 정리하자면 Relay는 기본적으로 Subject와 비슷한 특성을 가지지만 onCompIte 및 onError가 없기 때문에 끊임없이 데이터를 emit 할 수 있다는 점만 알아두자.