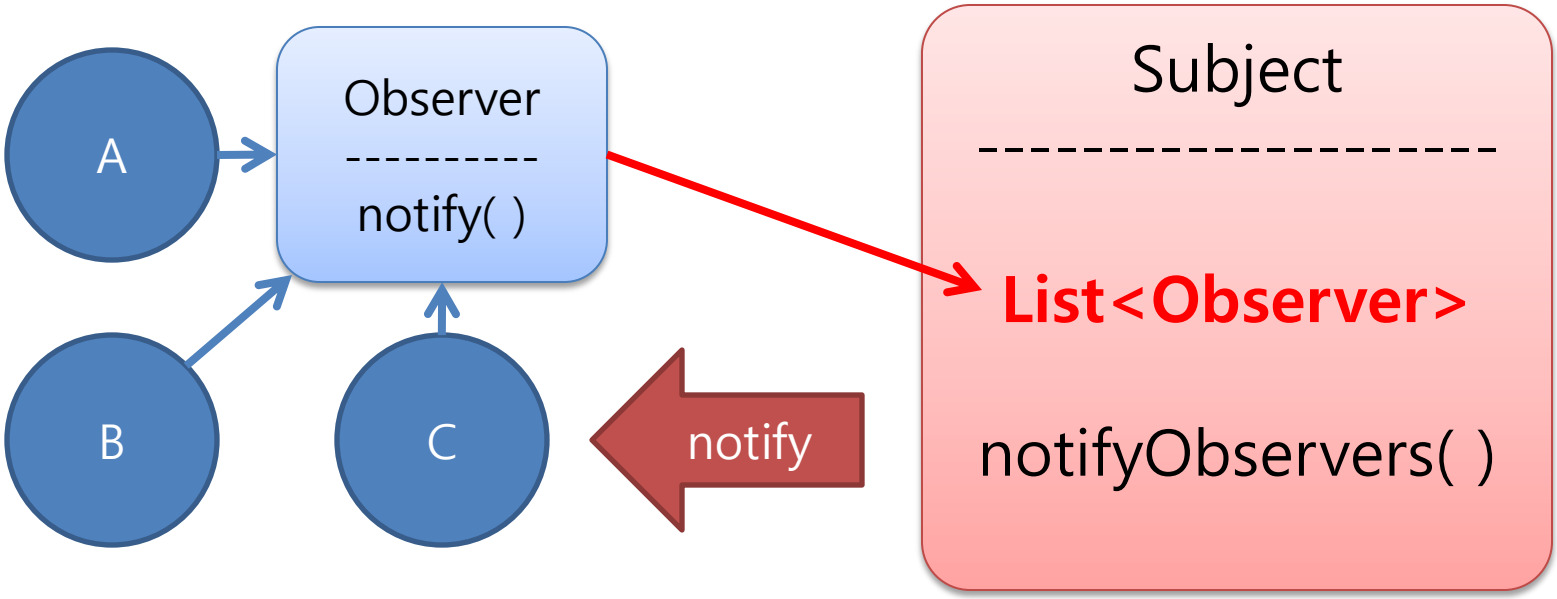
Android
035 RX Java

1. Observer Pattern

1. 대상이 되는 객체를 1대 n의 의존성으로 묶을 수 있는 방법을 제공하고 대상 객체에 변경사항이 있으면 의존성이 있는 모든 객체에 자동으로 알림을 제공하는 디자인 패턴이다

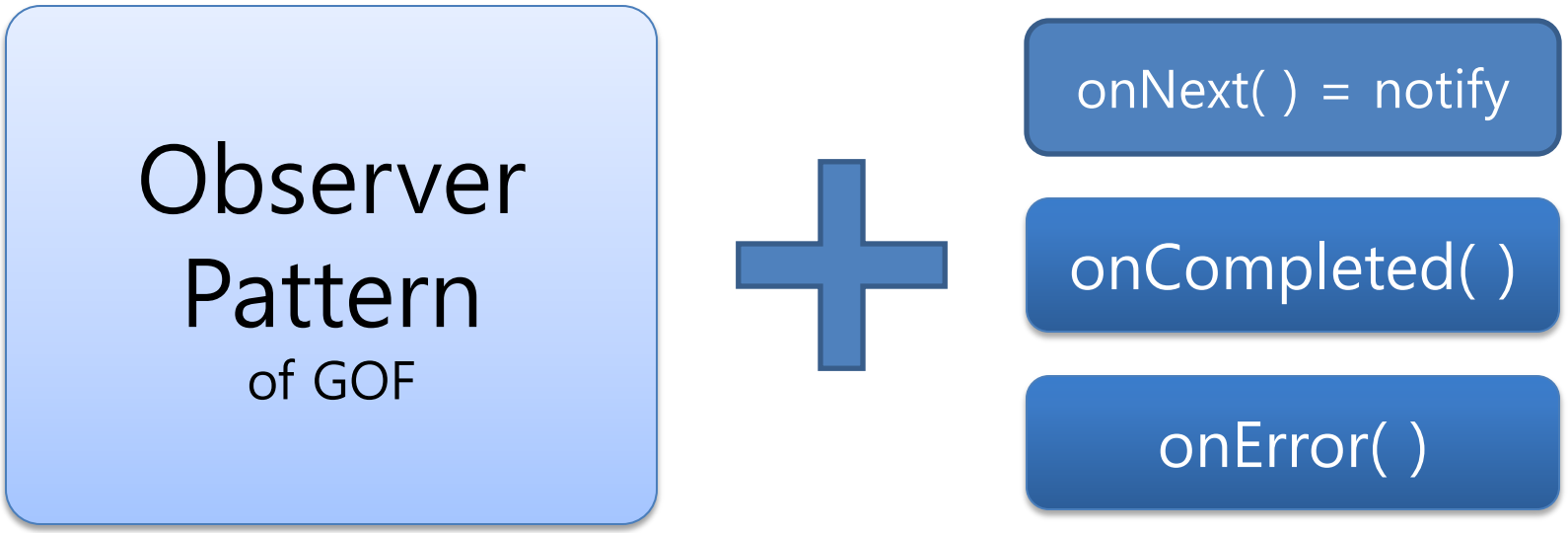
2. Observer 패턴은 Subject 에 기반을 두고 Subject 에 변경사항이 있으면 자신의 List에 존재하는 Observer 들에게 알림을 전달한다



2. Observable of RxJava

1. Gof 의 Observer 패턴에서 확장된 디자인 패턴

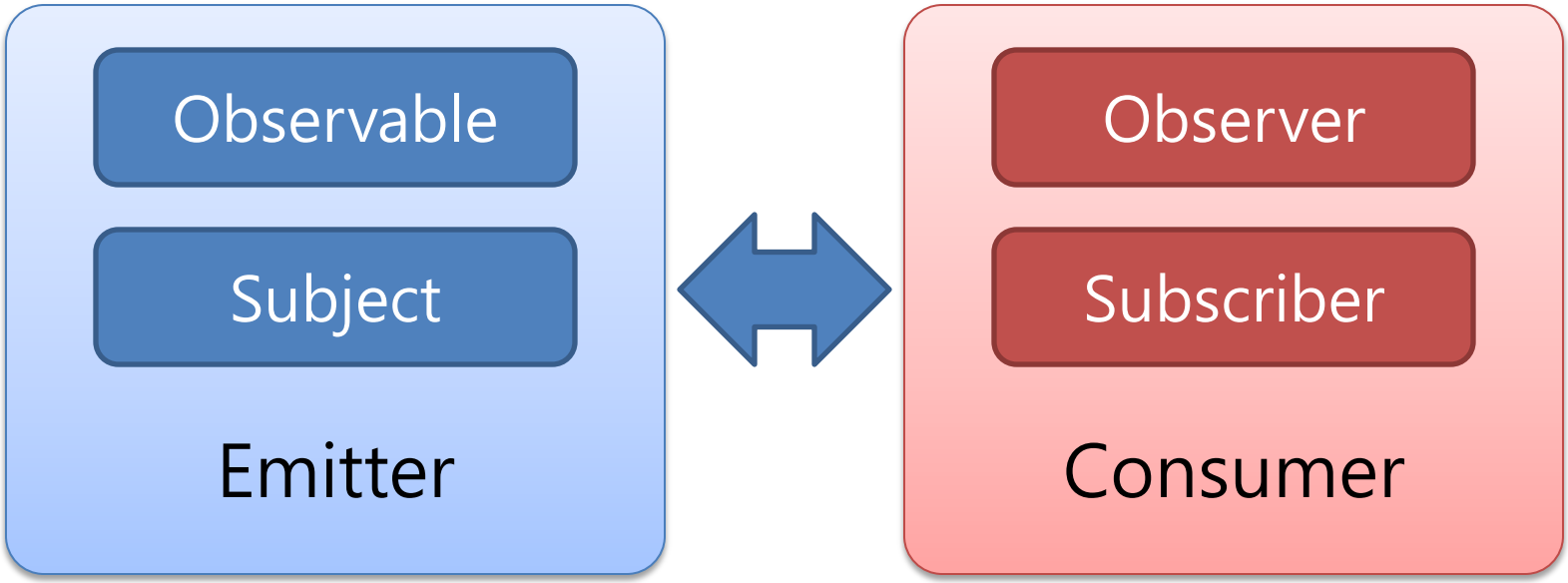
2. 아래 두 가지 새로운 기능 추가
 - onCompleted( )
   더 이상 전달할 데이터가 없음을 Observer 에게 알린다
 - onError( )
   에러가 발생했음을 Observer 에게 알린다



3. RxJava

- RxJava의 Main Players

Observable은 아이템들을 발행(emit)하고, **Subscribers는 아이템들을 소비(consume)**한다. Observer로 어떤 이벤트를 Subscriber에 보내면 Subscriber는 그 이벤트들을 처리하는 방식으로 동작한다. 또 Subscriber는 Observable에서 보낸 아이템을 onNext()에서 처리하는데 처리가 끝나면 onComplete를 호출하고 중간에 에러가 발생하면 onError()를 호출하게된다.



4. Iterable vs Observable of RxJava

1. Observable 은 3개의 생명주기 이벤트를 가지고 있다

onNext( )	다음 데이터를 가져온다
onError( )	에러가 발생했다
onCompleted( )	데이터 처리가 완료되었다

2. 이는 Iterable 의 이벤트와 유사하지만 데이터 흐름이 각각 Pull 과 Push 형태로 서로 다르게 동작한다

Event	Observable	Iterable
get value	onNext(T)	T next( )
exception	onError(T)	throws Exception
finish task	onCompleted( )	not hasNext( )
after finish task	Push values	Do nothing

5. Hot and Cold Observable of RxJava

1. Hot Observable

값이 생성되자마자 발행을 시작한다

이 Observable 을 구독하는 Observer는 값의 발행 중간(특정되지않음)에서부터 시퀀스를 받아보게 된다

2. Cold Observable

값이 생성된 후 Observer가 구독할 때 까지 대기한다

모든 값에 대한 시퀀스를 보장받을 수 있다

3. Observable 생성 - Operators

Observable.create( ) : 새로생성

.from( ) : 이미 생성된 리스트로 부터 시퀀스 생성

.just( ) : 자바객체를 Observable 로 변환하고자 할 때

.empty( ) : 아무것도 발행하지 않으면서 정상적으로 종료

.never( ) : 아무것도 발행하지 않고 종료도 되지 않음

.throw( ) : 아무것도 발행하지 않고 에러와 함께 종료

(* Rxjava의 error(Throwable t) 과 같음)

6. Operators of RxJava

1. Operator

- Observable 을 생성하는 명령어
- 각 Operator는 생성, 변형, 분류, 조합, 에러처리, 유틸리티, 조건과 상태(boolean), 수학과 집계, backpressure, connectable, 변환으로 분류 된다

2. 생성 Operator

- create : 모든 스트림은 선언되는 순간 메모리에 할당된다 (defer 제외)
- just
- from
- defer : 데이터스트림이 subscribe가 호출 될 때에 메모리에 할당된다
- empty
- throw
- never
- interval : 특정 시간간격으로 아이템 발행
- range : 범위값만큼 아이템을 발행하며 횟수의 제한이 있음
- start : 함수의 결과값 발행. RxJavaAsyncUtil을 디펜던시에 추가해야 한다
- repeat : 일정수만큼 반복
- timer : 일정시간 이후에 단일항목을 발행

7. Subject = Observable + Observer

1. PublishSubject
- 기본적인 Subject

2. BehaviorSubject
- 가장 최근에 관찰된 아이템과 그 후에 관찰된 나머지 아이템을 Observer에게 발행

3. ReplaySubject
- 관찰한 모든 아이템을 버퍼에 저장하고 Observer에게 재생

4. AsyncSubject
- Observable이 완료됐을 때 구독하고 있는 각 Observer에게 관찰한 마지막 아이템만을 발행

8.1. Hello RxAndroid !

1. build.gradle (app) 설정 1

- lambda 사용을 위해 java8 사용 설정을 추가한다

```
android {
    defaultConfig {
        applicationId "com.veryworks.android.study.rxjava_basic"
        ...
        // java 8 사용설정 1
        jackOptions {
            enabled true
        }
    }
    // java 8 사용설정 2
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```


8.2. Hello RxAndroid !

2. build.gradle (app) 설정 2

- rxandroid 추가 및 기타 라이브러리 추가

```
dependencies {

    ...

    compile 'io.reactivex:rxandroid:1.2.1'

    compile 'com.android.support:recyclerview-v7:25.+'
    compile 'com.android.support:cardview-v7:25.+'
    compile 'com.google.code.gson:gson:2.3'

    ...

}
```


9.1. RX Java Scheduler

Schedulers.computation() - 이벤트 그룹에서 간단한 연산이나 콜백 처리를 위해 사용. RxComputationThreadPool라는 별도의 스레드 풀에서 최대 cpu갯수 만큼 순환하면서 실행
Schedulers.immediate() - 현재 스레드에서 즉시 수행. observeOn()이 여러 번 쓰였을 경우 immediate()를 선언한 바로 윗쪽의 스레드에서 실행.
Schedulers.from(executor) - 특정 executor를 스케쥴러로 사용.

Schedulers.io() - 동기 I/O를 별도로 처리시켜 비동기 효율을 얻기 위한 스케줄러. 자체적인 스레드 풀 CachedThreadPool을 사용. API 호출 등 네트워크를 사용한 호출 시 사용.
Schedulers.newThread() - 새로운 스레드를 만드는 스케쥴러
Schedulers.trampoline() - 큐에 있는 일이 끝나면 이어서 현재 스레드에서 수행하는 스케쥴러.

AndroidSchedulers.mainThread() - 안드로이드의 UI 스레드에서 동작
HandlerScheduler.from(handler) - 특정 핸들러 handler에 의존하여 동작

9.2. Scheduler 사용 주의점

Schedulers.computation() - 이벤트 룹에서 간단한 연산이나 콜백 처리를 위해서 사용. I/O 처리를 여기에서 해서는 안됨.
Schedulers.io() - 동기 I/O를 별도로 처리시켜 비동기 효율을 얻기 위한 스케줄러. 자체적인 스레드 풀에 의존.

일부 오퍼레이터들은 자체적으로 어떤 스케쥴러를 사용할지 지정한다. 예를 들어 **buffer** 오퍼레이터는 Schedulers.computation()에 의존하며 **repeat**은 Schedulers.trampoline()를 사용한다.

- RxAndroid 지정 Scheduler
AndroidSchedulers.mainThread() - 안드로이드의 UI 스레드에서 동작.
HandlerScheduler.from(handler) - 특정 핸들러 handler에 의존하여 동작.

- RxAndroid가 제공하는 AndroidSchedulers.mainThread() 와 **RxJava가 제공하는 Schedulers.io()를 조합**해서 Schedulers.io()에서 수행한 결과를 AndroidSchedulers.mainThread()에서 받아 UI에 반영하는 형태가 많이 사용되고 있다

99. RX Java를 쓰지 말아야 할때

- When Not to Use RxJava
http://tomstechnicalblog.blogspot.kr/2016/07/when-not-to-use-rxjava.html

Case #1 Small, Constant, and Unchanging Data Sets
- 크기가 작고 상수이거나 변하지 않는 데이터 셋

Case #2 Expensive, Cached Objects
- 정규표현식 필드를 사용하여 인스턴스를 만드는 비용이 클 경우

Case #3 Simple "Lookups" and Single-Step Monads
- 간단한 '찾기'나 한 단계 조작만이 필요한 경우

Case #4 Frequently Qualified Properties
- 빈번하게 필터링(filtered/qualified)되는 필드

Case #5 Capturing State
- 상태를 저장할 필요가 있는 경우

