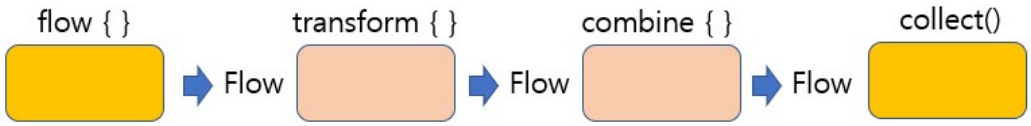
01-8. Flow Intermediate operator

Coroutine

Intermediate opterator

- Intermediate operator 란 Flow 의 확장함수
- Flow 에 의해 발행되는 데이터를 다양하게 핸들링하는 방법을 제공해 주는 함수



map(), filter(), take()

- map() 은 Flow 에 의해 발행되는 데이터를 map { } 을 거쳐 구독되게 하기 위한 operator
- filter() 는 filter { } 에서 true 를 리턴한 데이터만 이용하게 하기 위한 operator
- take() 는 개수를 지정하여 그 개수까지의 데이터만 이용하기 위한 operator

flowOn()

- Flow 내부에서 이용되는 CoroutineContext 는 Flow 를 실행시킨 곳의 CoroutineContext 을 그대로 이용해야 하며 이를 Context Preservation 이라 한다.

- Flow Builder, Flow Intermediator 는 항상 Collector 의 Context 내에서 실행

flowOn()

- flowOn() 은 Flow 를 실행시킨 CoroutineContext 를 교체하기 위해서 사용되는 operator

```
•flow { ... }
    .map { ... }          } 이 부분은 Dispatchers.IO에서 실행
    .flowOn(Dispatchers.IO)
    .filter { ... }       } 이 부분은 호출자 컨텍스트에서 실행
    .collect { ... }      }
```

stateIn(), shareIn()

- stateIn 과 shareIn 은 Flow 를 StateFlow 혹은 SharedFlow 로 바꾸기 위한 operator
- Flow 는 Cold Stream 이며 StateFlow, SharedFlow 는 Hot Stream
- stateIn, shareIn 을 이용해 Flow 의 Cold Stream 을 StateFlow 혹은 SharedFlow 을 이용하는 Hot Stream 으로 변경해 사용하고자 할 때 이용

stateIn()

```
fun <T> Flow<T>.stateIn(
    scope: CoroutineScope,
    started: SharingStarted,
    initialValue: T
): StateFlow<T>
```

- stateIn 은 cold Stream 을 Hot Stream 으로 변경하기 위한 operator 이며 반환 타입은 StateFlow
- 초기값을 initialValue 로 지정

stateIn()

- flowOn() , stateIn() 모두 새로운 코루틴을 만들고 Context 지정 가능
- flowOn() 은 성능 향상을 위해 사용하는 intermediator operator
- stateIn() 은 상태 발행이 주 목적인 terminal operator

측면	flowOn	stateIn
주된 목적	컨텍스트 변경	Flow → StateFlow 변환
연산자 타입	중간 연산자	터미널 연산자
코루틴 생성 시점	collect 시	호출 즉시 (정책에 따라)
적용 범위	상류(upstream)만	전체 Flow

stateIn()

- started 매개변수 값은 어떻게 데이터를 발행해야 하는지를 명시
  - Eagerly : 구독자가 없다고 하더라도 Hot Stream 이 만들어지면 바로 실행되어 데이터가 발행
  - Lazily : 첫번째 구독자가 존재해야 데이터를 발행하기 시작
  - WhileSubscribed() : 구독자가 존재하는 경우에만 데이터를 발행하며, 함수의 매개변수 설정 값으로 구독자가 사라진 후 어떻게 움직일 것인지를 지정

- WhileSubscribed() 함수의 매개변수에 설정되는 값
  - stopTimeoutMillis : 구독자가 모두 사라진 후 데이터 발행을 정지할 시간을 지정. 0 이면 바로 정지
  - replayExpirationMillis : 구독자가 모두 사라진 이후 replay 에 데이터를 유지해야 하는 시간

shareIn()

```
fun <T> Flow<T>.shareIn(
    scope: CoroutineScope,
    started: SharingStarted,
    replay: Int = 0
): SharedFlow<T>
```

- replay 는 새로운 구독자가 발생했을 때 전달해야 하는 이전 발행 값의 개수
- 기본 값은 0이며 0보다 큰 수를 지정하면 그 수만큼의 버퍼가 설정되고 새로운 구독자를 위해 발행한 데이터를 버퍼에 저장

transform()

- transform() 은 Flow 에 의해 발행되는 데이터를 받아 새로운 데이터를 발행하고자 할 때 사용
- map(), filter(), take() 등의 다른 operator 와 다르게 transform { } 의 { } 에 지정한 코드는 FlowCollector 에서 실행
- transform { } 내에서 emit() 을 이용해 데이터 발행이 가능한 operator

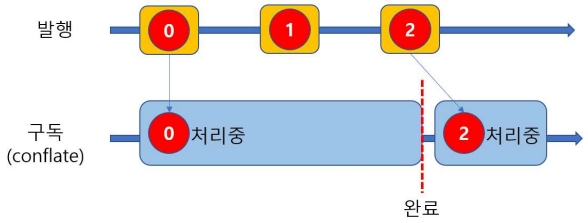
buffer()

- 일반적으로 Flow 는 데이터 발행과 구독이 순차적으로 진행
- 데이터를 생산하는 속도가 데이터를 소비하는 속도보다 빠른 경우 미리 데이터를 발행해 버퍼에 담아 놓은 것이 효율적일 수 있는데 이때 사용하는 operator 가 buffer()

- buffer() 를 이용하면서 capacity 매개변수를 이용해 버퍼 사이즈를 지정
- onBufferOverflow 매개변수를 SUSPEND, DRAP_OLDEST, LATEST 중 하나를 지정하여 버퍼가 가득 찬 경우 새로 발행하는 데이터를 어떻게 해야할지 지정

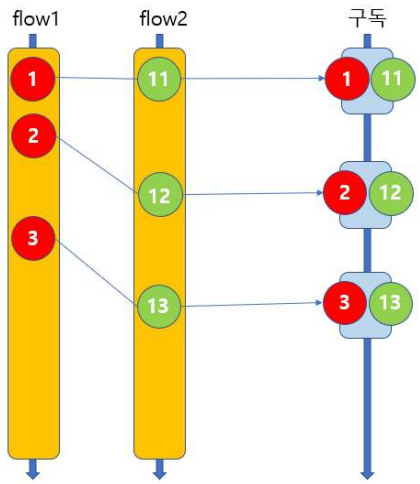
conflate()

- 데이터가 여러 번 발생한다고 하더라도 구독 쪽이 실행되는 시점의 최신데이터만 이용되게 하고 싶은 경우 사용하는 operator



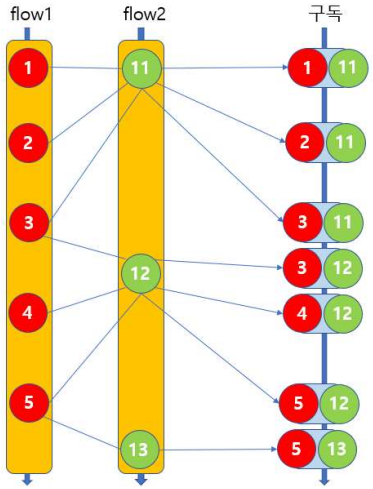
zip()

- zip() 은 두 flow 에서 발행하는 데이터를 같이 구독하기 위해 사용되는 operator
- zip() 에 의해 데이터가 발행되는 시점은 두 flow 의 데이터가 모두 새로 발행된 시점



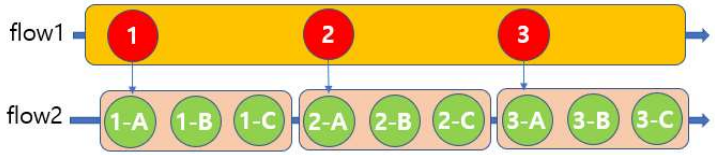
combine()

- combine() 도 zip() 과 마찮가지로 두 flow 의 발행 데이터를 같이 받기위해 사용
- combine() 은 어느 쪽 flow 든 새로운 데이터가 발행되기만 하면 combine() 에 의해 그 순간의 최신 데이터가 발행



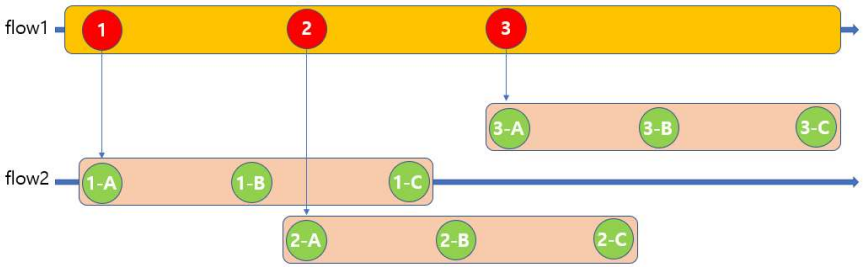
flatMapConcat()

- flatMapConcat() 와 flatMapMerge() 는 두 flow 를 연결해 하나의 flow에서 발행한 데이터에 의해 다른 flow 가 실행되게 할 때 사용
- flatMapConcat() 은 이중 루프와 흡사



flatMapConcat()

- flatMapMerge() 는 flatMapConcat() 과 목적은 동일합니다. 두 flow 를 연결하고 하나의 flow 에서 발행한 데이터에 의해 다른 flow 를 실행시키는 operator
- . flow1 과 flow2 가 있다고 가정하고 flow1에서 발행하는 데이터에 의해 flow2가 실행된다고 가정해보면 flatMapConcat() 은 순차적으로 flow2 가 실행되지만 flatMapMerge() 는 병렬적으로 flow2 가 실행



catch()

- catch() 는 Flow 에서 발생하는 예외를 처리하기 위한 operator
- catch() operator 영역은 FlowCollector 에서 동작하게 됨으로 Flow 에서 발생한 예외를 적절하게 처리한 후 다시 emit() 을 이용해 데이터를 발행 가능

onCompletion()

- onCompletion() 은 데이터 구독이 모두 완료된 후에 실행되는 operator
- Flow 에서 데이터가 발행이 완료되거나 아니면 예외가 발생해서 구독이 종료되는 경우에도 실행

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare