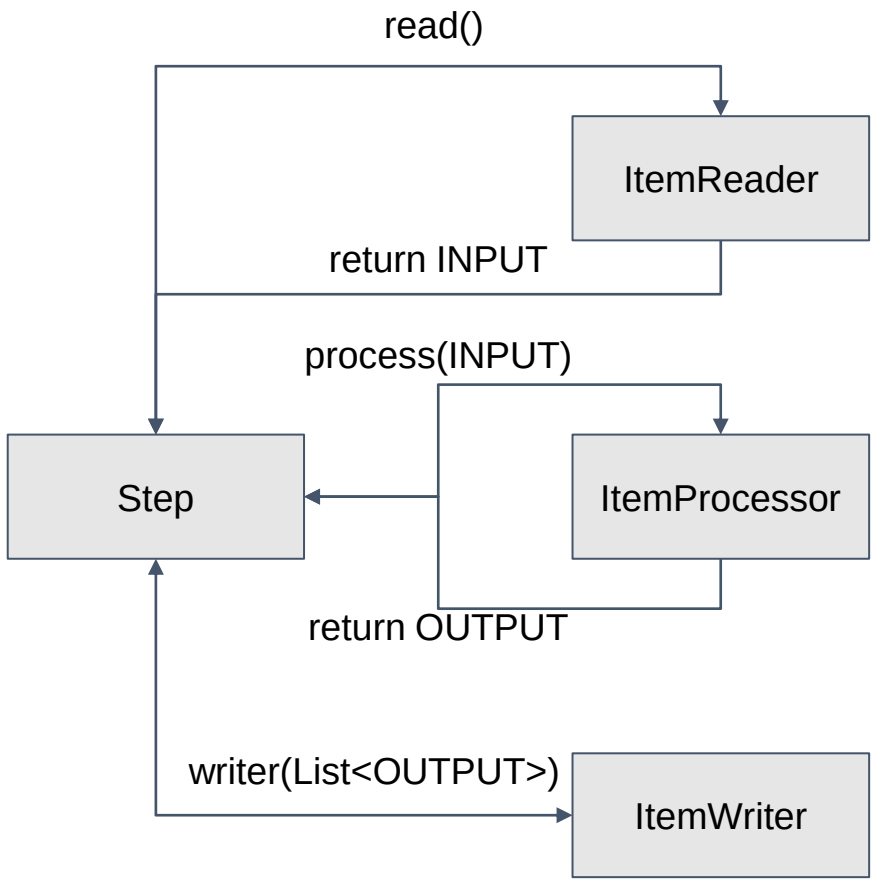
Chapter 03. 스프링 배치 기초 이해하기

Spring Batch

- 배치를 처리할 수 있는 방법은 크게 2가지
- Tasklet을 사용한 Task 기반 처리
  - 배치 처리 과정이 비교적 쉬운 경우 쉽게 사용
  - 대량 처리를 하는 경우 더 복잡
  - 하나의 큰 덩어리를 여러 덩어리로 나누어 처리하기 부적합
- Chunk를 사용한 chunk(덩어리) 기반 처리
  - ItemReader, ItemProcessor, ItemWriter의 관계 이해 필요
  - 대량 처리를 하는 경우 Tasklet 보다 비교적 쉽게 구현
  - 예를 들면 10,000개의 데이터 중 1,000개씩 10개의 덩어리로 수행
    - 이를 Tasklet으로 처리하면 10,000개를 한번에 처리하거나, 수동으로 1,000개씩 분할
- 예제 참고

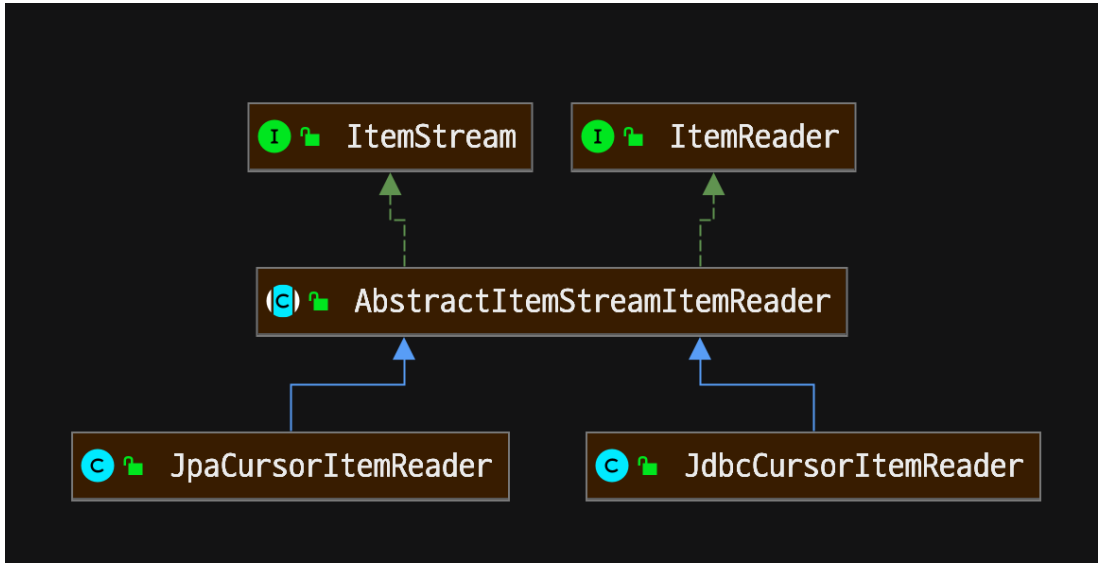


- reader에서 null을 return 할 때 까지 Step은 반복
- **<INPUT, OUTPUT>chunk(int)**
  - reader에서 INPUT 을 return
  - processor에서 INPUT을 받아 processing 후 OUPUT을 return
    - INPUT, OUTPUT은 같은 타입일 수 있음
  - writer에서 List<OUTPUT>을 받아 write

- 배치를 실행에 필요한 값을 parameter를 통해 외부에서 주입
- JobParameters는 외부에서 주입된 parameter를 관리하는 객체
- parameter를 **JobParameters**와 **Spring EL(Expression Language)**로 접근
  - String parameter = jobParameters.getString(key, defaultValue);
  - @Value(“#{jobParameters[key]}”)
- 예제 참고

- @Scope는 어떤 시점에 bean을 생성/소멸 시킬 지 bean의 lifecycle을 설정
- @JobScope는 job 실행 시점에 생성/소멸
  - Step에 선언
- @StepScope는 step 실행 시점에 생성/소멸
  - Tasklet, Chunk(ItemReader, ItemProcessor, ItemWriter) 에 선언
- Spring의 @Scope과 같은 것
  - @Scope(“job”) == @JobScope
  - @Scope(“step”) == @StepScope
- Job과 Step 라이프사이클에 의해 생성되기 때문에 Thread safe하게 작동
- @Value(“#{jobParameters[key]}”)를 사용하기 위해 @JobScope와 @StepScope는 필수

- 배치 대상 데이터를 읽기 위한 설정
  - 파일, DB, 네트워크, 등에서 읽기 위함.
- Step에 ItemReader는 필수
- 기본 제공되는 ItemReader 구현체
  - file, jdbc, jpa, hibernate, kafka, etc...
- ItemReader 구현체가 없으면 직접 개발
- ItemStream은 ExecutionContext로 read, write 정보를 저장
- CustomItemReader 예제 참고



- FlatFileItemReader 클래스로 파일에 저장된 데이터를 읽어 객체에 매핑
- 예제 참고

- Cursor 기반 조회
  - 배치 처리가 완료될 때 까지 DB Connection이 연결
  - DB Connection 빈도가 낮아 성능이 좋은 반면, 긴 Connection 유지 시간 필요
  - 하나의 Connection에서 처리되기 때문에, Thread Safe 하지 않음
  - 모든 결과를 메모리에 할당하기 때문에, 더 많은 메모리를 사용
- Paging 기반 조회
  - 페이징 단위로 DB Connection을 연결
  - DB Connection 빈도가 높아 비교적 성능이 낮은 반면, 짧은 Connection 유지 시간 필요
  - 매번 Connection을 하기 때문에 Thread Safe
  - 페이징 단위의 결과만 메모리에 할당하기 때문에, 비교적 더 적은 메모리를 사용

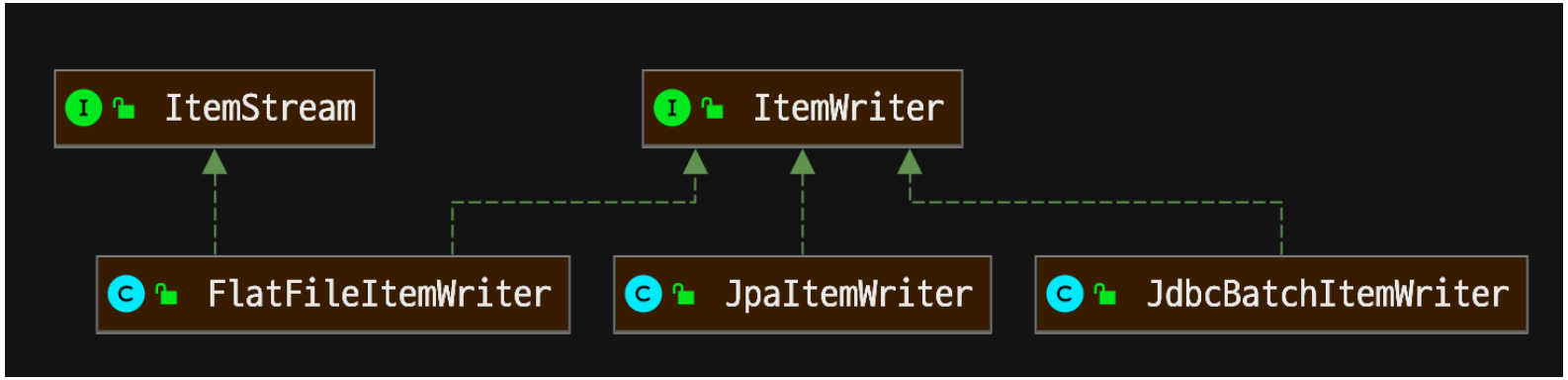
- JdbcCursorItemReader 예제 참고

	JdbcCursorItemReader	JdbcPagingItemReader
datasource	JDBC를 실행하기 위한 Datasource	
beanMapper | rowMapper	조회된 데이터 row를 클래스와 매핑하기 위한 설정	
sql	조회 쿼리 설정	X
selectClause, fromClause, whereClause 또는 queryProvider	X	조회 쿼리 설정
fetchSize	cursor에서 fetch될 size	JdbcTemplate.fetchSize
pageSize	X	paging에 사용될 page 크기(offset/limit)

- 4.3+ 에서 Jpa 기반 Cursor ItemReader가 제공됨.
- 기존에는 Jpa는 Paging 기반의 ItemReader만 제공됨.

	JpaCursorItemReader	JpaPagingItemReader
entityManagerFactory	JPA를 실행하기 위해 EntityManager를 생성하기 위한 EntityManagerFactory	
	조회된 데이터 row를 클래스와 매핑하기 위한 설정	
queryString	조회 쿼리	X
selectClause, fromClause, whereClause	X	조회 쿼리
pageSize	X	paging에 사용될 page 크기(offset/limit)

- ItemWriter는 마지막으로 배치 처리 대상 데이터를 어떻게 처리할 지 결정
- Step에서 ItemWriter는 필수
- 예를 들면 ItemReader에서 읽은 데이터를
  - DB에 저장, API로 서버에 요청, 파일에 데이터를 write
- 항상 write가 아님
  - 데이터를 최종 마무리를 하는 것이 ItemWriter



- FlatFileItemWriter는 데이터가 매핑된 객체를 파일로 write
- 예제 참고

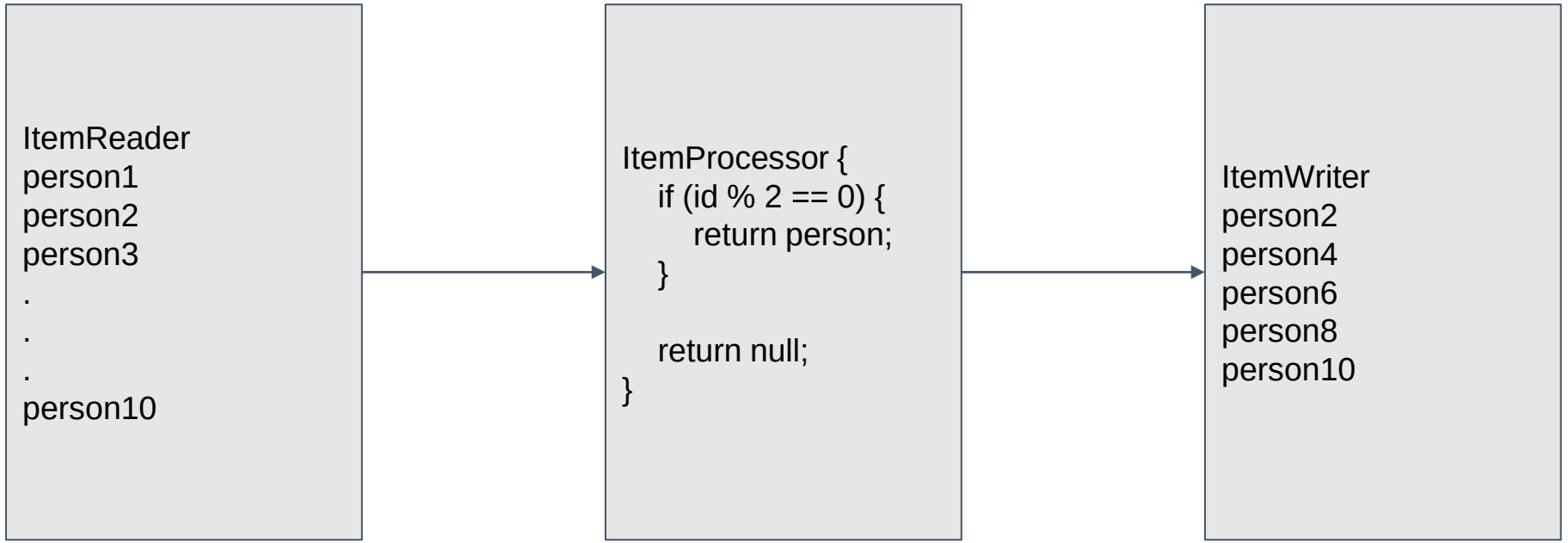
- JdbcBatchItemWriter는 jdbc를 사용해 DB에 write
- JdbcBatchItemWriter는 bulk insert/update/delete처리
  - insert into person (name, age, address) values (1,2,3), (4,5,6), (7,8,9);
- 단건 처리가 아니기 때문에 비교적 높은 성능
- 예제 참고

- JpaItemWriter는 JPA Entity 기반으로 데이터를 DB에 write
- Entity를 하나씩 EntityManager.persist 또는 EntityManager.merge로 insert
- 예제 참고

- ItemReader에서 읽은 데이터를 가공 또는 Filtering
- Step의 ItemProcessor는 optional
- ItemProcessor는 필수는 아니지만, 책임 분리를 분리하기 위해 사용
- ItemProcessor는 I(input)를 O(output)로 변환하거나
- ItemWriter의 실행 여부를 판단 할 수 있도록 filtering 역할을 한다.
    - ItemWriter는 not null만 처리한다.

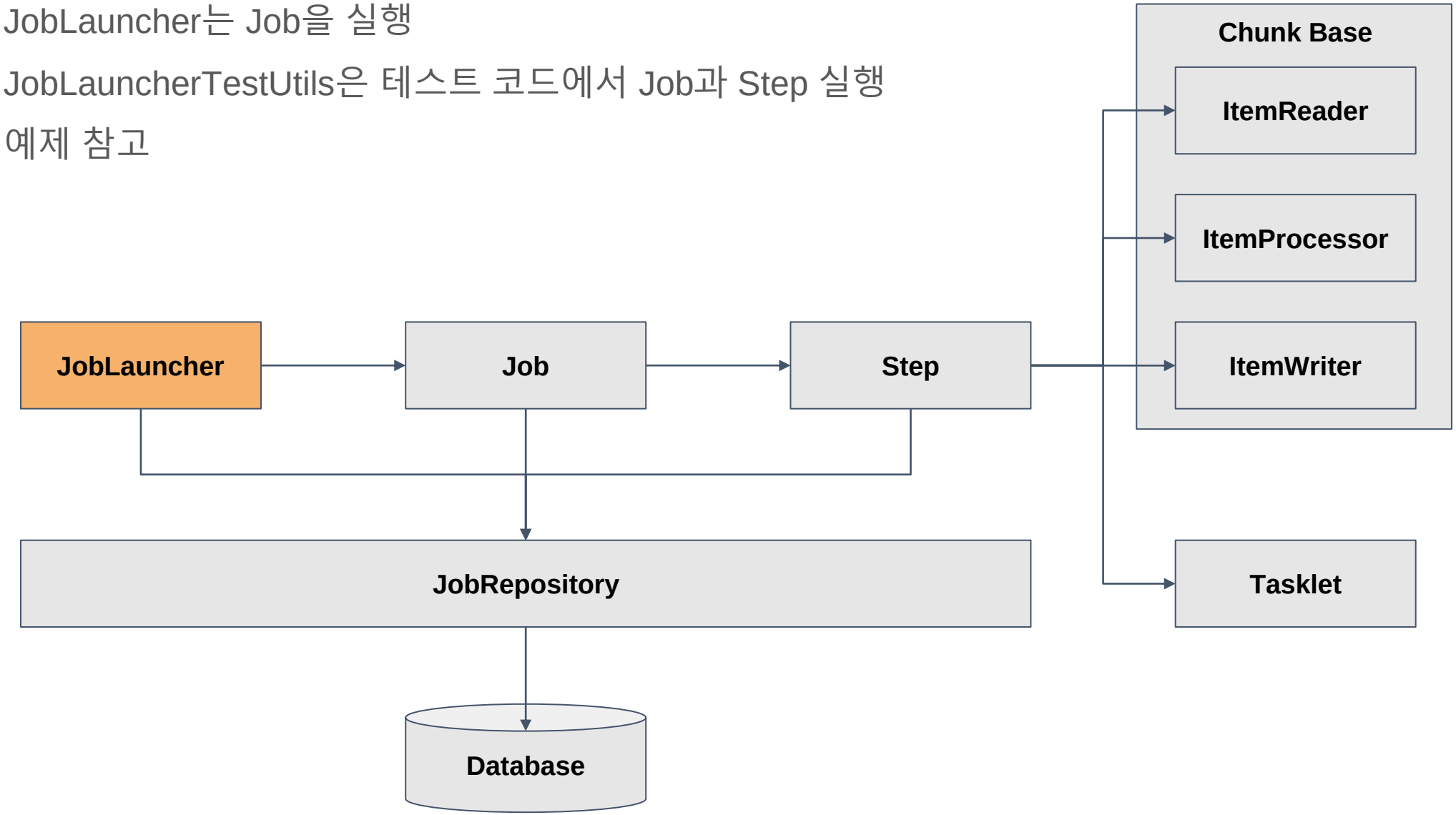
```java
public interface ItemProcessor<I, O> {

    Process the provided item, returning a potentially modified or new item for
    continued processing. If the returned result is null, it is assumed that
    processing of the item should not continue. A null item will never reach
    this method because the only possible sources are:
    • an ItemReader (which indicates no more items)
    • a previous ItemProcessor in a composite processor (which indicates a
      filtered item)

    Params:  item – to be processed, never null.
    Returns: potentially modified or new item for continued processing, null if
             processing of the provided item should not continue.
    Throws:  Exception – thrown if exception occurs during processing.

    @Nullable
    O process(@NonNull I item) throws Exception;
}
```

- 예를 들어 person.id가 짝수인 person만 return 하는 경우
- ItemWriter는 5개의 person만 받아 처리
- 예제 참고

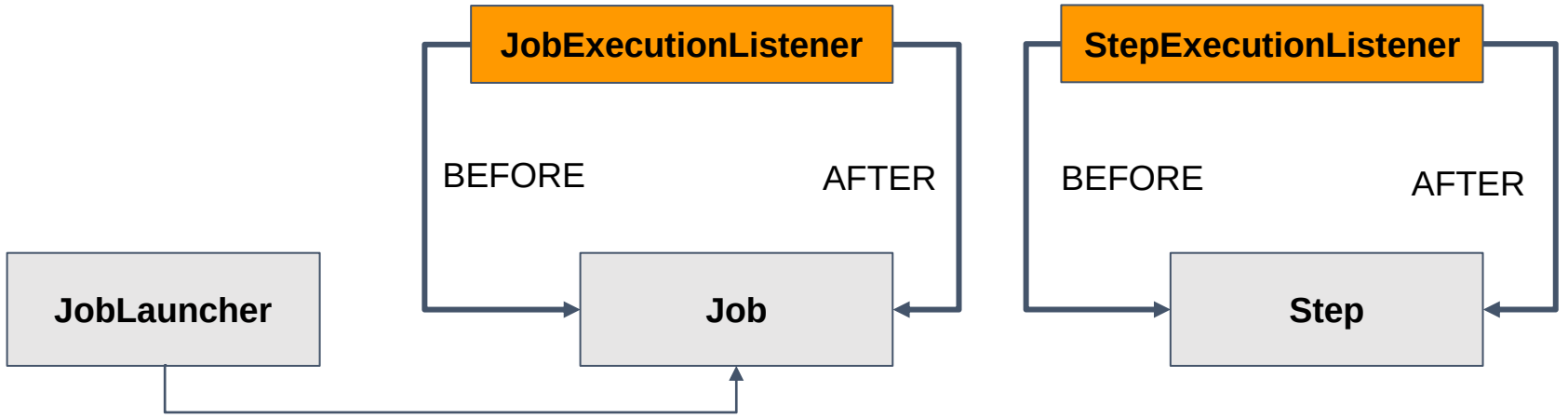


- CSV 파일 데이터를 읽어 H2 DB에 데이터 저장하는 배치 개발
- Reader
  - 100개의 person data를 csv 파일에서 읽는다.
- Processor
  - allow_duplicate 파라미터로 person.name의 중복 여부 조건을 판단한다.
  - `allow_duplicate=true` 인 경우 모든 person을 return 한다.
  - `allow_duplicate=false 또는 null` 인 경우 person.name이 중복된 데이터는 null로 return 한다.
  - 힌트 : 중복 체크는 `java.util.Map` 사용
- Writer
  - 2개의 ItemWriter를 사용해서 Person H2 DB에 저장 후 몇 건 저장됐는 지 log를 찍는다.
  - Person 저장 ItemWriter와 log 출력 ItemWriter
  - 힌트 : `CompositeItemWriter` 사용

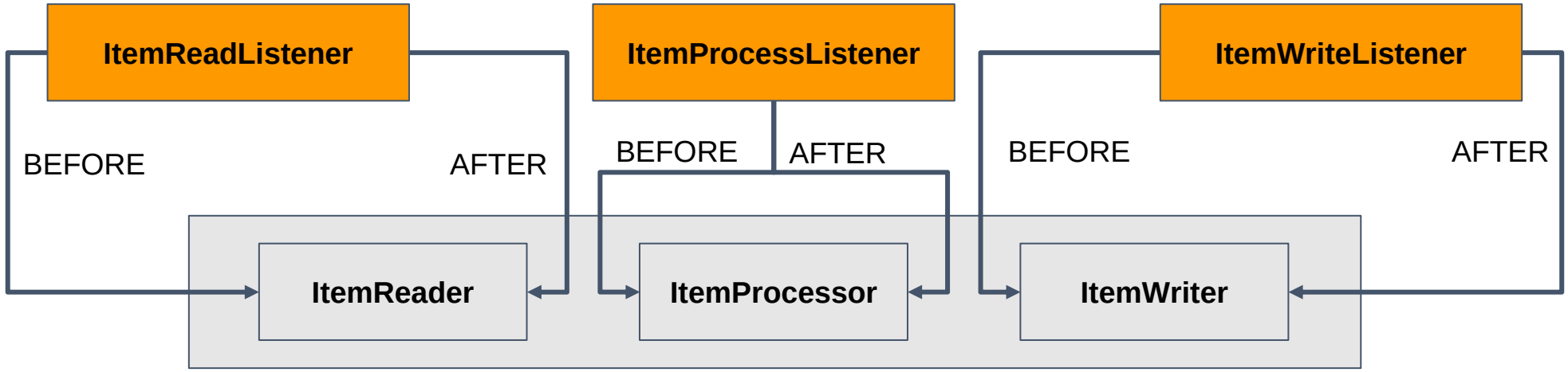
- JobLauncher는 Job을 실행
- JobLauncherTestUtils은 테스트 코드에서 Job과 Step 실행
- 예제 참고



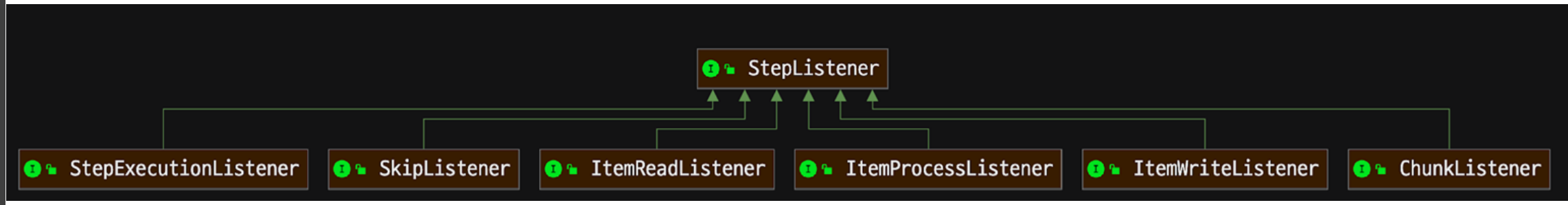
- 스프링 배치에서 **전 처리**, **후 처리**를 하는 다양한 종류의 Listener 존재.
  - interface 구현
  - @Annotation 정의
- Job 실행 전과 후에 실행할 수 있는 JobExecutionListener
- Step 실행 전과 후에 실행할 수 있는 StepExecutionListener



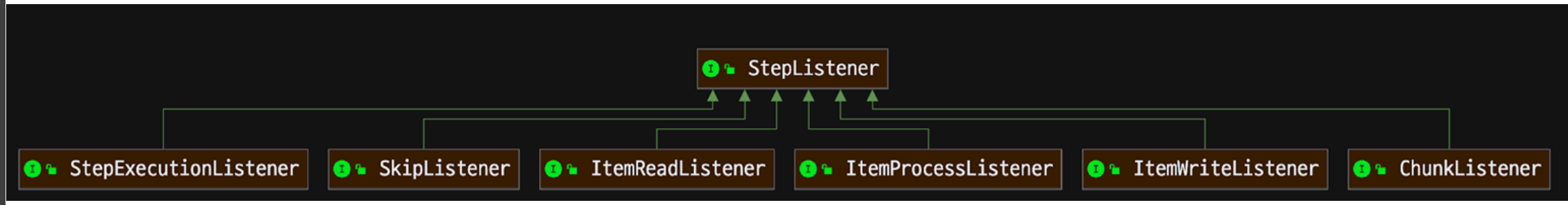
- ItemReader 전, 후, 에러 처리 ItemReadListener
- ItemProcessor 전, 후, 에러 처리 ItemProcessListener
- ItemWriter 전, 후, 에러 처리 ItemWriteListener



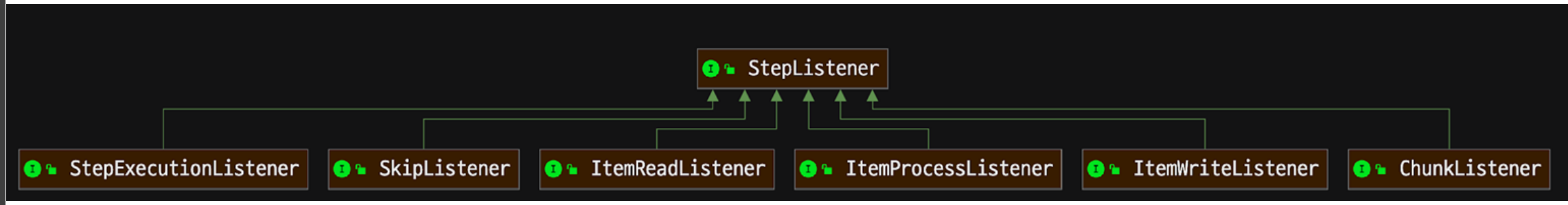
- Step에 관련된 모든 Listener는 StepListener를 상속
- StepExecutionListener
- SkipListener
- ItemReadListener
- ItemProcessListener
- ItemWriteListener
- ChunkListener



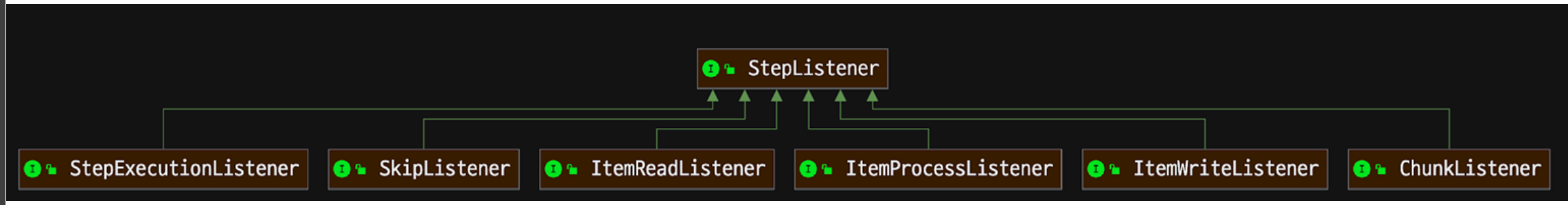
- SkipListener
  - onSkipInRead : @OnSkipInRead
    - ItemReader에서 Skip이 발생한 경우 호출
  - onSkipInWrite : @OnSkipInWrite
    - ItemWriter에서 Skip이 발생한 경우 호출
  - onSkipInProcess : @OnSkipInProcess
    - ItemProcessor에서 Skip이 발생한 경우 호출



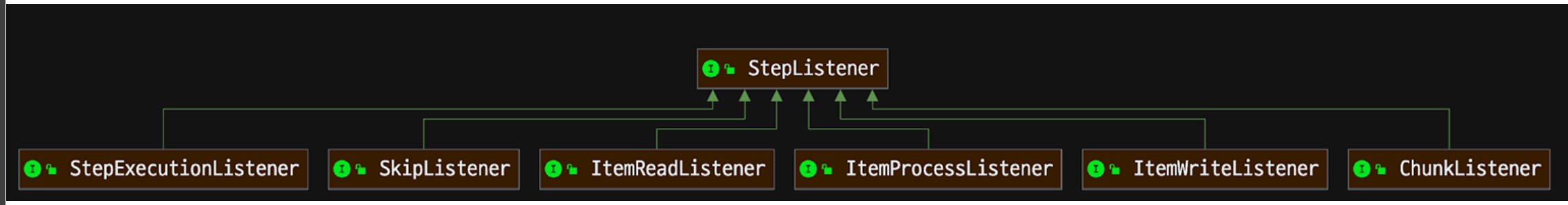
- ItemReadListener
  - beforeRead : @BeforeRead
    - ItemReader.read() 메소드 호출 전 호출
  - afterRead : @AfterRead
    - ItemReader.read() 메소드 호출 후 호출
  - onReadError : @OnReadError
    - ItemReader.read() 메소드에서 에러 발생 시 호출



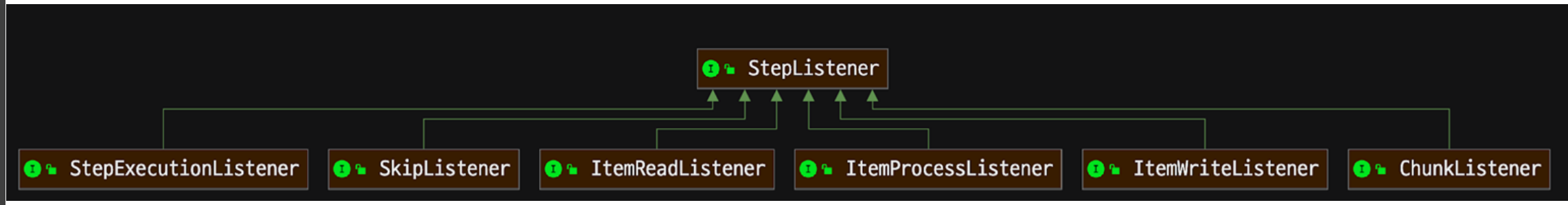
- ItemWriteListener
  - beforeWrite : @BeforeRead
    - ItemWriter.write() 메소드 호출 전 호출
  - afterWrite : @AfterRead
    - ItemWriter.write() 메소드 호출 후 호출
  - onWriteError : @OnWriteError
    - ItemWriter.write() 메소드에서 에러 발생 시 호출



- ItemProcessListener
  - beforeProcess : @BeforeProcess
    - ItemProcess.process() 메소드 호출 전 호출
  - afterProcess : @AfterProcess
    - ItemProcess.process() 메소드 호출 후 호출
  - onProcessError : @OnProcessError
    - ItemProcess.process() 메소드에서 에러 발생 시 호출



- ChunkListener
  - beforeChunk : @BeforeChunk
    - chunk 실행 전 호출
  - afterChunk : @BfterChunk
    - chunk 실행 후 호출
  - afterChunkError : @BfterChunkError
    - chunk 실행 중 에러 발생 시 호출



- step 수행 중 발생한 특정 Exception과 에러 횟수 설정으로 예외처리 설정
- skip(NotFoundNameException.class), skipLimit(3) 으로 설정된 경우
  - NotFoundNameException 발생 3번까지는 에러를 skip 한다.
  - NotFoundNameException 발생 4번째는 Job과 Step의 상태는 실패로 끝나며, 배치가 중지된다.
  - 단, 에러가 발생하기 전까지 데이터는 모두 처리된 상태로 남는다.
- Step은 chunk 1개 기준으로 Transaction 동작
  - 예를 들어 items = 100, chunk.size =10, 총 chunk 동작 횟수 = 10
    - chunk 1-9는 정상 처리, chunk 10에서 Exception이 발생한 경우
    - chunk 1-9 에서 처리된 데이터는 정상 저장되고, Job과 Step의 상태는 FAILED 처리
  - 배치 재 실행 시 chunk 10 부터 처리할 수 있도록 배치를 만든다.

- 추가 요구사항
  - Person.name이 empty String인 경우 NotFoundNameException 발생
  - NotFoundNameException이 3번 이상 발생한 경우 step 실패 처리
- SkipListener가 실행 되는 조건
  - 에러 발생 횟수가 skipLimit 이하인 경우
    - skipLimit(2), throw Exception이 3번 발생하면 실행되지 않는다.
    - skipLimit(3), throw Exception이 3번 발생하면 실행된다.
    - skip 설정 조건에 해당하는 경우에만 실행된다.
  - SkipListener는 항상 faultTolerant() 메소드 후에 선언

- Step 수행 중 간헐적으로 Exception 발생 시 재시도(retry) 설정
  - DB Deadlock, Network timeout 등
- retry(NullPointerException.class), retryLimit(3) 으로 설정된 경우
  - NotFoundNameExeception이 발생한 경우 3번까지 재시도
- 더 구체적으로 retry를 정의하려면 RetryTemplate 이용
- 추가 요구사항
  - NotFoundNameException이 발생하면, 3번 재 시도 후 Person.name을 “UNKNOWN” 으로 변경

1. RetryListener.open
    a. return false 인 경우 retry를 시도하지 않음.
2. RetryTemplate.RetryCallback
3. RetryListener.onError
    a. maxAttempts 설정값 만큼 반복
4. RetrtyTemplate.RecoveryCallback
    a. maxAttempts 반복 후에도 에러가 발생한 경우 실행
5. RetryListener.close