

# 스프링 부트 웹 프로젝트

chapter13

## Spring Batch

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

# Contents

part.1

스프링 배치란?

part.2

배치의 구조

part.3

배치 사용하기

part.4

스케줄링 사용하기

스프링 배치(Spring Batch)는 대용량 데이터를 처리하기 위한 프레임워크이다.  
대량의 데이터를 처리하거나, 주기적이고 반복적인 작업을 실행하는 데 사용된다.  
또한, 배치의 상태를 추적하는 기능도 제공합니다.

## Spring Batch 5.1.2

### Batch

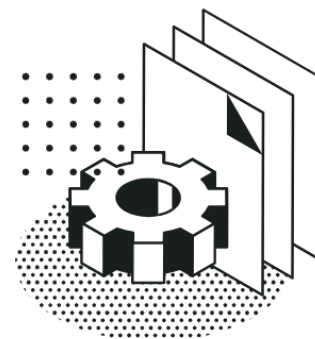
The ability of batch processing to efficiently process large amounts of data makes it ideal for many use cases. Spring Batch's implementation of industry-standard processing patterns lets you build robust batch jobs on the JVM. Adding Spring Boot and other components from the Spring portfolio lets you build mission-critical batch applications.

#### What is batch processing?

Batch processing is the processing of a finite amount of data in a manner that does not require external interaction or interruption.

#### Why build batch processes?

Batch processes are an extremely efficient way of processing large amounts of data. The ability to schedule and prioritize work based on SLAs lets you allocate resources for best utilization.



## 스프링 배치

## Spring Batch를 사용하는 이유

1. 정기적인 데이터 생성: 매일 또는 매월 정해진 시간에 데이터를 모아 통계나 일 매출을 집계하는 경우
2. 데이터 마이그레이션: 기존 시스템에서 새로운 새로운 시스템으로 대량의 데이터를 이관하는 경우
3. 대량 로그 분석: 서버 로그처럼 많은 데이터를 주기적으로 분석하고 집계하는 경우

차원		지표			
간단 상세		요약			
날짜	요일	결제			환불금액
		결제금액	모바일비율 (결제금액)	결제당 결제금액	
전체	전체	9,181,300	69%	41,172	
2019-04-30	화	1,600	100%	40,533	
2019-04-29	월	3,300	56%	47,413	
2019-04-28	일	5,600	100%	43,300	
2019-04-27	토	0,100	100%	30,033	
2019-04-26	금	4,700	65%	57,350	
2019-04-25	목	0,500	37%	43,813	
2019-04-24	수	4,900	100%	44,363	
2019-04-23	화	2,900	74%	50,414	
2019-04-22	월	3,700	86%	28,744	
2019-04-21	일	3,800	100%	51,255	
2019-04-20	토	1,400	100%	48,280	
2019-04-19	금	3,000	62%	36,917	

일별 판매 금액

## Spring Batch vs Quartz

스프링 배치는 대량의 데이터를 처리를 하기 위해 만들어진 프레임워크이다.

데이터 처리를 배치 작업으로 나누어 실행하며, 스케줄링 기능은 제공하지 않는다.

Quartz는 스케줄링 기능을 제공하는 도구로, 스프링 배치와 함께 사용할 수 있다.

Spring Boot / Reference / IO / Quartz Scheduler

## Quartz Scheduler

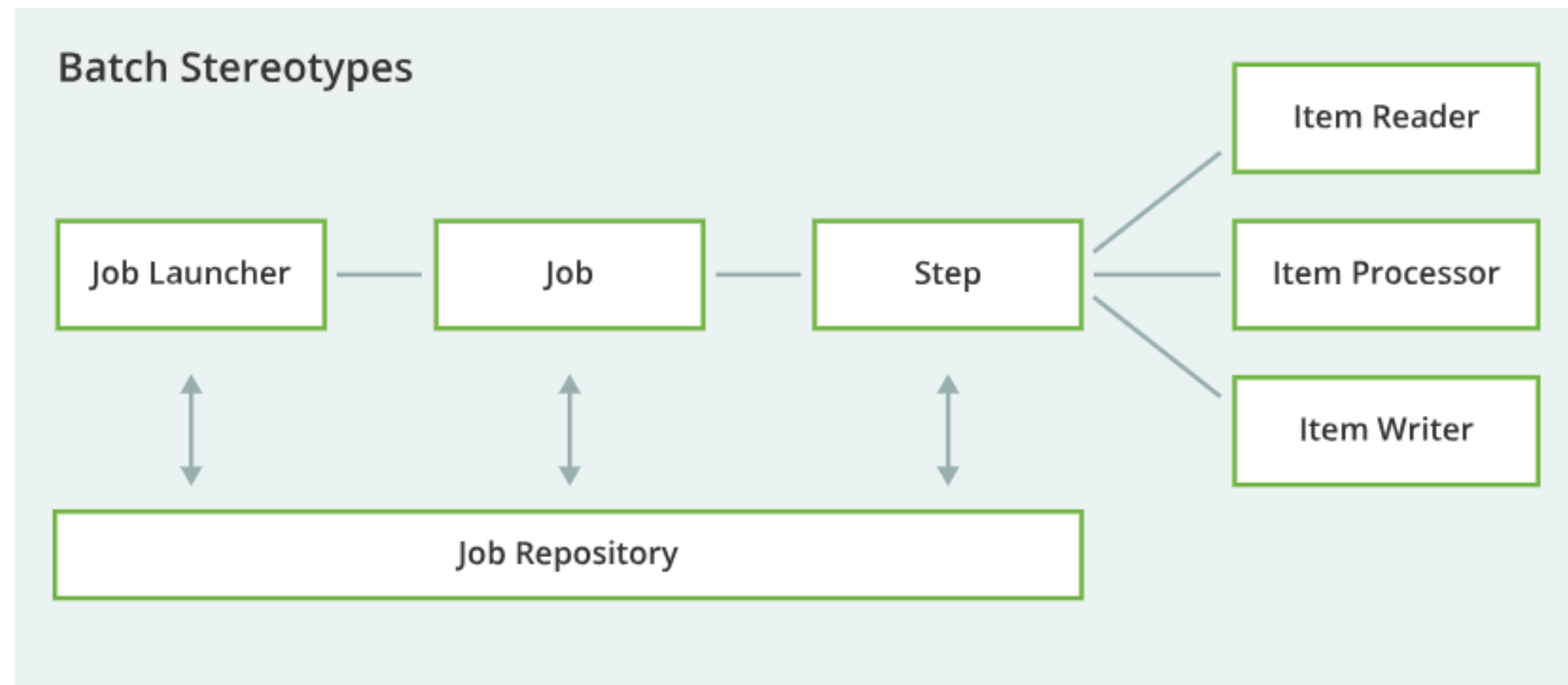
Spring Boot offers several conveniences for working with the [Quartz scheduler](#), including the `spring-boot-starter-quartz` starter. If Qu available, a `Scheduler` is auto-configured (through the `SchedulerFactoryBean` abstraction).

Beans of the following types are automatically picked up and associated with the `Scheduler`:

# 배치의 구조

## Spring Batch의 동작 방식

스프링 배치는 Job을 정의하여 대량의 데이터 처리 작업을 구성하며, Step으로 나누어 단계별로 작업을 수행한다. 각 Step은 ItemReader로 데이터를 읽고, ItemProcessor로 처리한 후, ItemWriter로 저장한다. JobRepository는 작업의 메타데이터와 상태를 관리하며, JobLauncher가 Job을 실행한다.



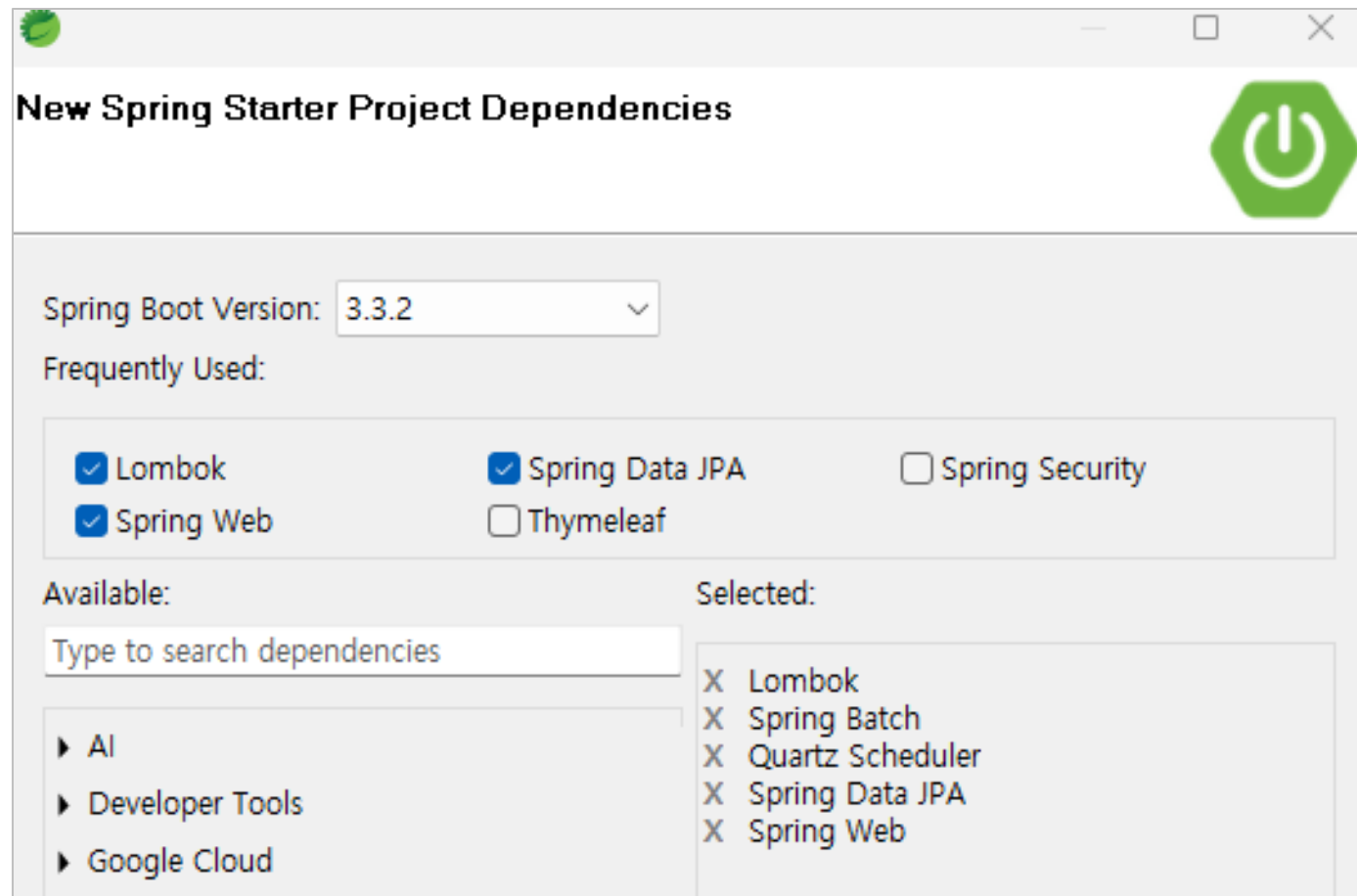
- JobLauncher: Job을 실행하는 컴포넌트로, Job의 실행을 시작하고 관리한다.  
실행 상태를 모니터링하고 필요한 설정을 제공한다.
- JobRepository: Job과 Step의 메타데이터를 저장하고 관리하는 저장소.  
Job의 실행 상태와 이력을 기록하여, 재실행이나 상태 복원을 지원한다.
- Job: 배치 작업의 단위로, 여러 개의 Step으로 구성된다.  
Job은 특정 목표를 달성하기 위해 정의된 작업의 전체적인 흐름을 나타낸다.
- Step: Job의 구성 요소로, 실제 데이터 처리 작업을 수행하는 단계.  
각 Step은 데이터 읽기, 처리, 쓰기 등의 작업을 포함할 수 있다.
- ItemReader: 데이터 소스에서 데이터를 읽어오는 역할. 데이터베이스, 파일, 큐 등 다양한 소스에서 데이터를 가져올 수 있다.
- ItemProcessor: 읽어온 데이터를 처리하거나 변환하는 단계. 데이터의 유효성 검사, 변환, 집계 등을 수행한다.
- ItemWriter: 처리된 데이터를 저장하거나 출력하는 역할. 데이터베이스에 저장하거나 파일로 작성하는 작업을 수행한다.
- Chunk: 데이터를 일정 크기로 나누어 처리하는 단위. 청크 처리를 사용하여 메모리 사용을 최적화하고 성능을 개선합니다.

배치 어플리케이션을 만들기 위해 Spring Batch와 Quartz를 함께 사용한다.

Spring Batch + Quartz

13번째 프로젝트를 생성한다.

라이브러리는 'Lombok, Spring Web, Spring Data JPA, Spring Batch, Quartz Scheduler'를 선택한다.





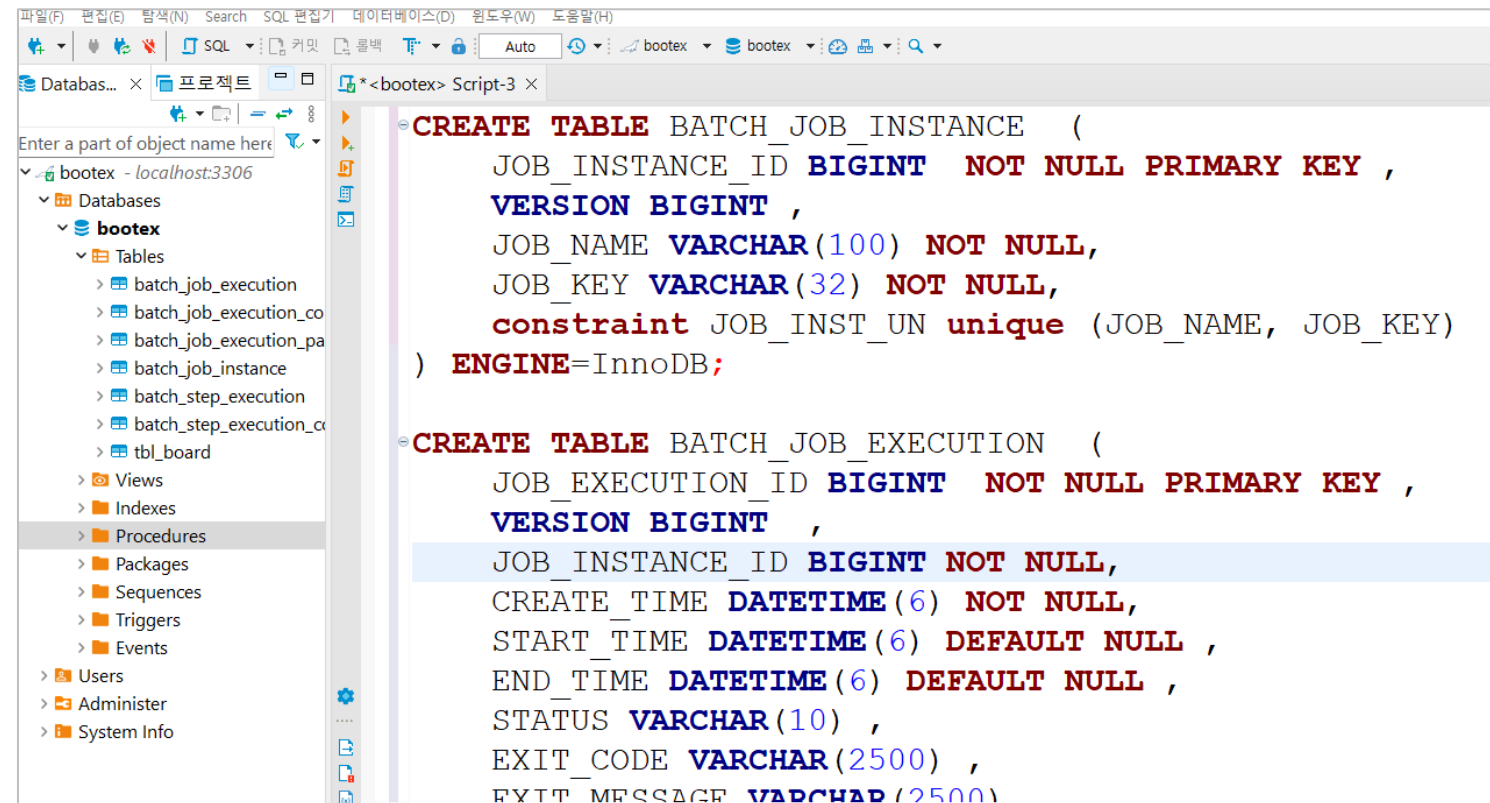
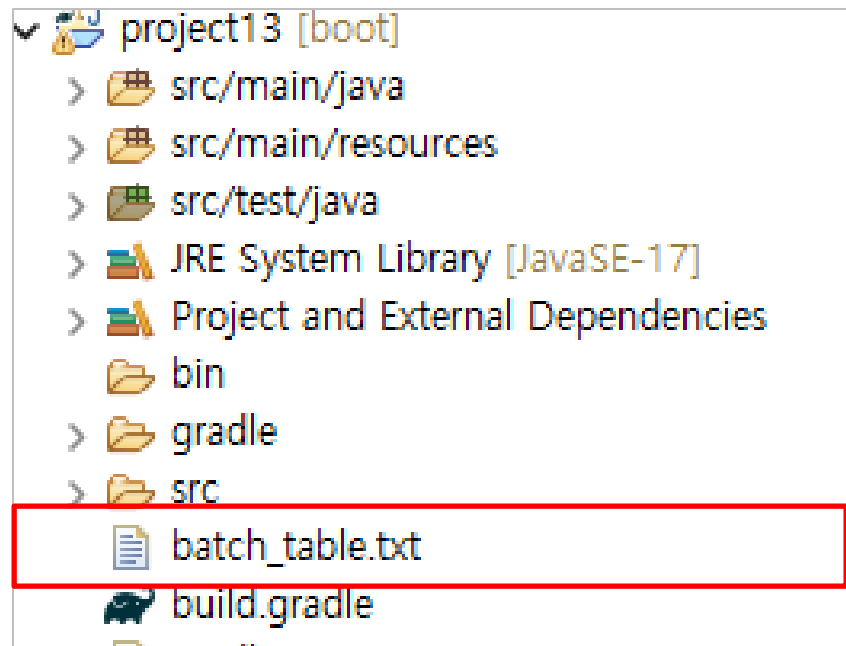
# 배치 사용하기

## 준비

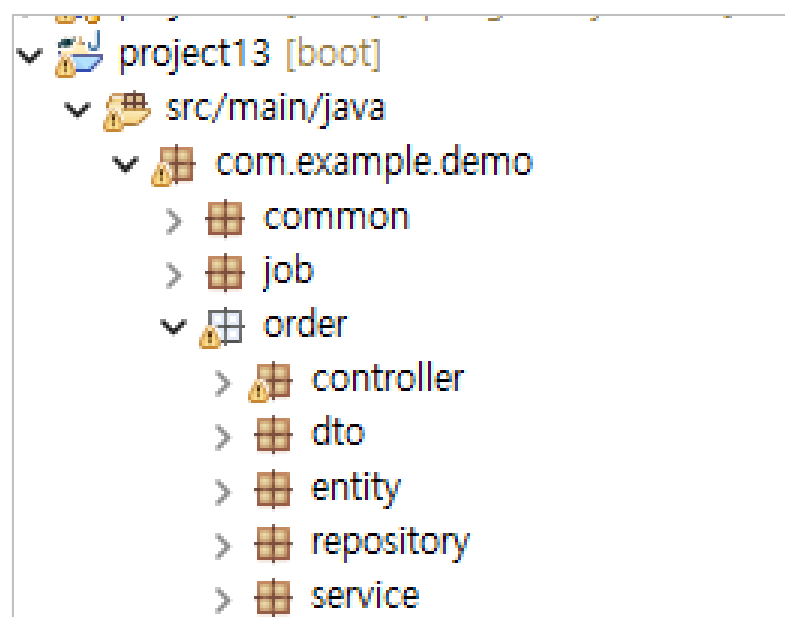
batch\_table.txt 파일을 열어서 내용을 복사한다.

디비버에서 SQL을 실행하여, 배치 작업에 필요한 테이블을 생성한다.

해당 테이블에는 배치가 수행될 때, 기록이 저장된다.



쇼핑몰에서 물건을 주문하는 기능과 주문이력을 통계로 계산하는 기능을 구현한다.



스프링 배치 작업을 설정하는 클래스를 만든다.

간단하게 작업과 스텝을 설정한다.

```
@Configuration
public class SimpleJobConfig {

    @Bean
    public Job simpleJob1() {
        return new JobBuilder("simpleJob", jobRepository)
            .start(step1())
            .next(step2())
            .build();
    }

    @Bean
    public Step step1() {
        return new StepBuilder("step1..", jobRepository)
            .tasklet(testTasklet(), platformTransactionManager).build();
    }

    @Bean
    public Step step2() {
        return new StepBuilder("step2..", jobRepository)
            .tasklet(test2Tasklet(), platformTransactionManager).build();
    }
}
```

# 배치 사용하기

## 배치 클래스 만들기

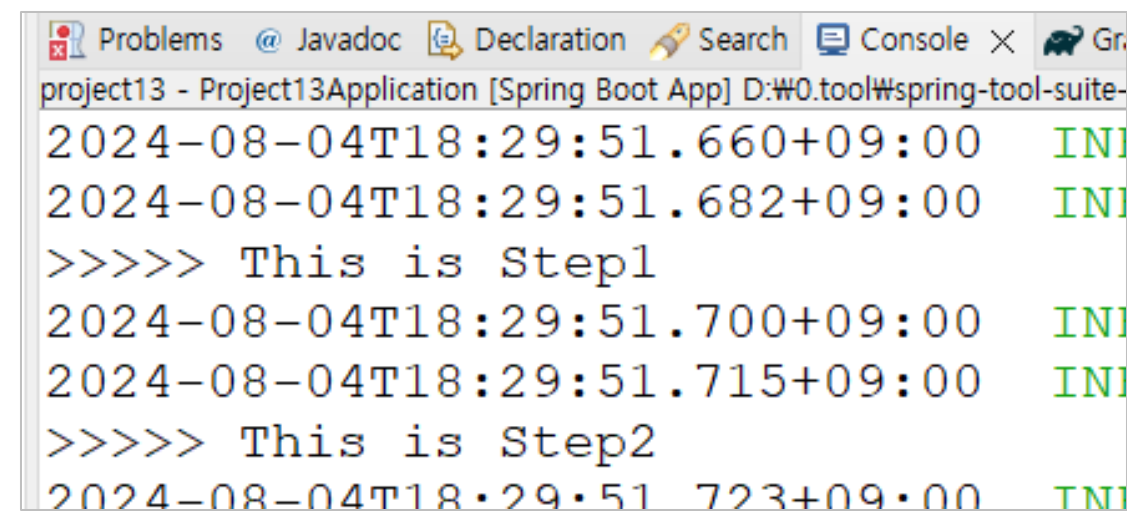
Tasklet은 배치 작업의 스텝을 구현하기 위한 인터페이스이다.

Tasklet는 함수형인터페이스로 Execute 메소드를 구현해야 한다.

작업이 완료되었는지 나타내는 상태를 반환한다.

```
@Bean
public Tasklet testTasklet() {
    return ((contribution, chunkContext) -> {
        System.out.println(">>>>> This is Step1");
        return RepeatStatus.FINISHED;
    });
}

@Bean
public Tasklet test2Tasklet() {
    return ((contribution, chunkContext) -> {
        System.out.println(">>>>> This is Step2");
        return RepeatStatus.FINISHED;
    });
}
```

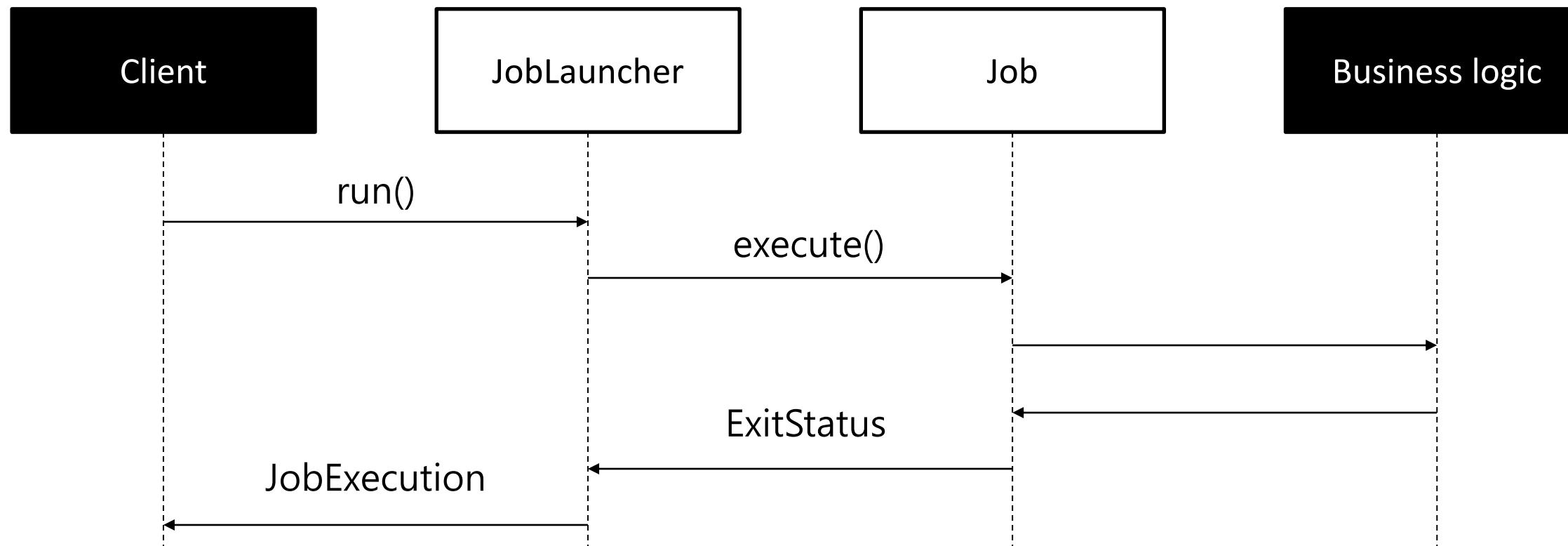


```
project13 - Project13Application [Spring Boot App] D:\W0\tool\spring-tool-suite-
2024-08-04T18:29:51.660+09:00    INI
2024-08-04T18:29:51.682+09:00    INI
>>>>> This is Step1
2024-08-04T18:29:51.700+09:00    INI
2024-08-04T18:29:51.715+09:00    INI
>>>>> This is Step2
2024-08-04T18:29:51.723+09:00    TNI
```

# 배치 사용하기

## 배치 클래스 만들기

Spring Boot Batch의 종류에는 Tasklet 방식, chunk 방식 등이 있다.  
Tasklet 방식은 Step 단계에서 하나의 작업만 처리하는 방식이다.  
단일 작업을 처리하기 때문에 작업이 끝날 때까지 대기한다.



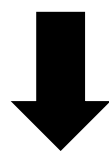
## 배치 사용하기

## 일별 주문 통계 구하기

쇼핑몰에서는 고객들의 주문을 매일 확인합니다.

이를 위해 일별로 주문 건수와 주문 금액을 계산해야 합니다.

오늘 날짜로 들어온 주문이력을 조회한다.  
총건수와 총금액을 계산한다.



통계 테이블에 오늘 날짜의 데이터를 등록한다.

	123 no	ABC product_name	123 price	mod_date	reg_date
1	1	무선 이어폰	75,000	2024-08-05 17:13:27.526	2024-08-03 17:13:27.526
2	2	블루투스 스피커	50,000	2024-08-05 17:13:32.153	2024-08-04 17:13:32.153
3	3	스마트 워치	120,000	2024-08-05 17:13:37.346	2024-08-05 17:13:37.346
4	4	노트북	1,500,000	2024-08-05 17:13:42.226	2024-08-05 17:13:42.226
5	5	게이밍 마우스	45,000	2024-08-05 17:13:45.859	2024-08-05 17:13:45.859

```

>>>>> This is Step1
OrderDTO(no=3, productName=스마트 워치, price=120000,
OrderDTO(no=4, productName=노트북, price=1500000, reg
OrderDTO(no=5, productName=게이밍 마우스, price=45000,

```

```

>>>>> This is Step2
총 건수:3
총 금액:1665000

```

	order_dt	123 count	mod_date
1	2024-08-05	3	2024-08-05 17:21:04.474

# 배치 사용하기

## 주문 이력 조회하고 집계하기

tasklet 함수에 비즈니스 로직을 추가한다.

오늘 들어온 주문 이력을 조회한 후에, 주문의 총 건수와 총금액을 계산한다.

다른 스텝과 공유하고 싶은 값을 job컨텍스트에 저장한다.

```
// Tasklet: 스텝에서 하나의 작업만 처리하는 방식
@Bean
public Tasklet testTasklet() {
    return ((contribution, chunkContext) -> {

        // 주문 이력 가져오기
        List<OrderDTO> list = orderService.getList();

        List<OrderDTO> filterList = list.stream()
            .filter(dto -> dto.getRegDate().toLocalDate().equals(LocalDate.now()))
            .collect(Collectors.toList());

        // 전체 건수와 총금액 구하기
        long count = filterList.stream().count();
        int totalPrice = filterList.stream().mapToInt(dto->dto.getPrice()).sum();

        // 공유 데이터 추가
        StepContext context = chunkContext.getStepContext();
        context.getStepExecution().getJobExecution().getExecutionContext().put("count", count);
        context.getStepExecution().getJobExecution().getExecutionContext().put("totalPrice", totalPrice);
    });
}
```

이전 스텝에서 계산한 값을 꺼내서, 집계 데이터를 만든다.

오늘 날짜의 집계 데이터를 등록한다.

```
@Bean
public Tasklet test2Tasklet() {
    return ((contribution, chunkContext) -> {

        System.out.println(">>>> This is Step2");

        // 공유 데이터 꺼내기
        StepContext context = chunkContext.getStepContext();
        Object count = context.getStepExecution().getJobExecution().getExecutionContext().get("count");
        Object totalPrice = context.getStepExecution().getJobExecution().getExecutionContext().get("totalPrice");

        // 집계 구하기
        int cnt = Integer.parseInt(count.toString());
        int total = Integer.parseInt(totalPrice.toString());

        // 집계 추가하기
        StatsDTO dto = StatsDTO.builder().orderDt(LocalDate.now()).count(cnt).totalPrice(total).build();
        statsService.register(dto);

        return RepeatStatus.FINISHED;
    });
}
```



# 배치 사용하기

# 메타 테이블 정보

batch\_job\_execution: 배치 작업의 실행 정보를 저장하는 테이블. 작업이 실행되고 끝나는 시간, 작업 결과가 저장됨.

batch\_job\_instance: 배치 작업 인스턴스를 저장하는 테이블. 작업의 이름과 식별자가 저장됨.

batch\_step\_execution: 배치 작업의 각 스텝 정보를 저장하는 테이블. 작업은 여러 개의 스텝으로 구성되며, 각 스텝의 실행정보가 저장됨. 스텝이 실행되고 끝나는 시간, 작업결과, 읽거나 쓰는 작업의 개수, 오류로 건너뛴 항목의 수가 저장됨.

batch\_job\_execution

	<a href="#">123</a> JOB_EXECUTION_ID ▾	<a href="#">123</a> VERSION ▾	<a href="#">123</a> JOB_INSTANCE_ID ▾	<a href="#">🕒</a> CREATE_TIME ▾	<a href="#">🕒</a> START_TIME ▾	<a href="#">🕒</a> END_TIME ▾	<a href="#">ABC</a> STATUS ▾	<a href="#">ABC</a> EXIT_CODE ▾	<a href="#">ABC</a> EXIT_MESSAGE ▾	<a href="#">🕒</a> LAST_UPDATED ▾
1	5	2	<a href="#">4</a> ↗	2024-08-05 17:21:04.264	2024-08-05 17:21:04.280	2024-08-05 17:21:04.500	COMPLETED	COMPLETED		2024-08-05 17:21:04.500

batch\_job\_instance

	<a href="#">123</a> JOB_INSTANCE_ID ▾	<a href="#">123</a> VERSION ▾	<a href="#">ABC</a> JOB_NAME ▾	<a href="#">ABC</a> JOB_KEY ▾
1	4	0	simpleJob	d41d8cd98f00b204e9800998ecf8427e

batch\_step\_execution

	<a href="#">123</a> STEP_EXECUTION_ID ▾	<a href="#">123</a> VERSION ▾	<a href="#">ABC</a> STEP_NAME ▾	<a href="#">123</a> JOB_EXECUTION_ID ▾	<a href="#">🕒</a> CREATE_TIME ▾	<a href="#">🕒</a> START_TIME ▾	<a href="#">🕒</a> END_TIME ▾	<a href="#">ABC</a> STATUS ▾	<a href="#">123</a> COMMIT_COUNT ▾	<a href="#">123</a> READ_COUNT ▾	<a href="#">123</a> FILTER_CC
1	7	3	step1..	<a href="#">5</a> ↗	2024-08-05 17:21:04.289	2024-08-05 17:21:04.291	2024-08-05 17:21:04.435	COMPLETED	1	0	
2	8	3	step2..	<a href="#">5</a> ↗	2024-08-05 17:21:04.442	2024-08-05 17:21:04.445	2024-08-05 17:21:04.496	COMPLETED	1	0	

# 스케줄링 사용하기

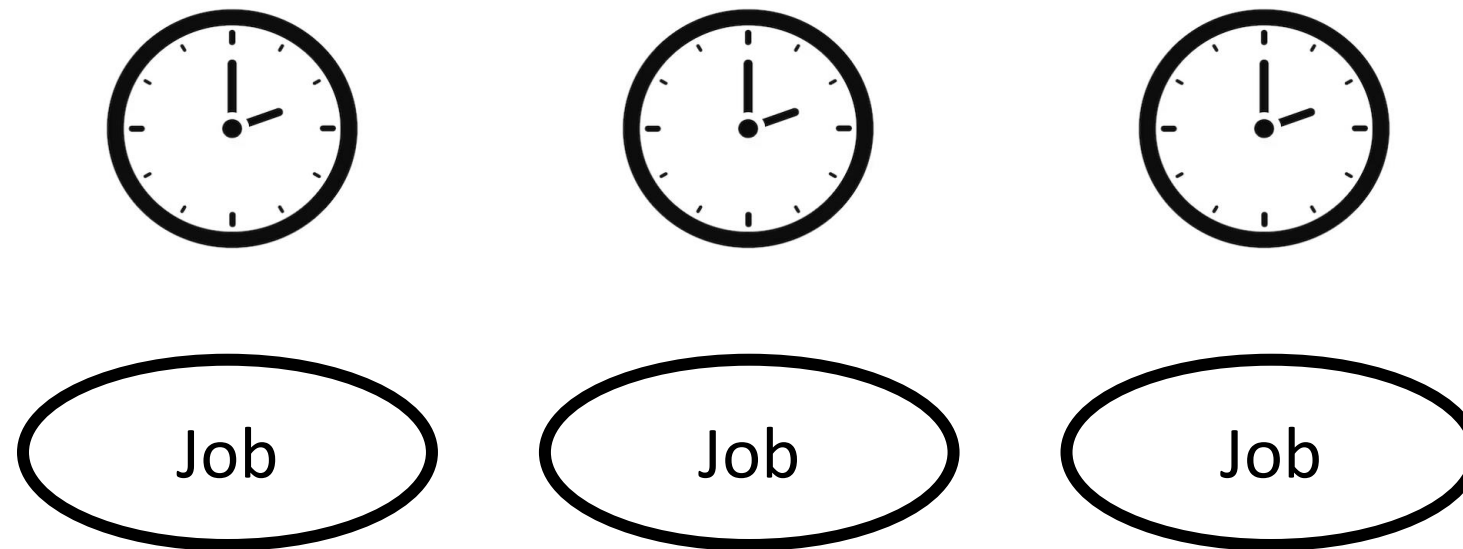
## 스케줄링이란?

### 스케줄링이란?

스케줄링은 매일, 매주, 매월 등 주기적으로 프로그램을 자동으로 실행하는 기능이다.

스케줄링 도구로는 리눅스의 CRON, 윈도우의 Task Scheduler, 스프링의 Quartz 등 여러가지가 있다.

Spring batch에는 스케줄링 기능이 없으므로, Quartz를 사용한다.



스케줄링 기능을 사용하기 위해 메인클래스에 @EnableScheduling 어노테이션을 추가해야 한다.

```
@EnableScheduling
@EnableJpaAuditing
@SpringBootApplication
public class Project13Application {

    public static void main(String[] args) {
        SpringApplication.run(Project13Applicatio
    }
}
```

# 스케줄링 사용하기

## 스케줄러 만들기

스케줄링이 정상적으로 동작하는지 확인하기 위해 간단한 코드를 작성한다.

매분 0초가 되면 "task run"을 출력한다.

@Scheduled(cron = "0 \* \* \* \* ") 이렇게 설정하면 매분 0초가 될 때마다 함수를 실행한다

```
@Component
public class SampleScheduler {

    // cron: 주기 (매분 0초가 될때마다 실행)
    @Scheduled(cron = "0 * * * * ")
    public void test() {
        System.out.println("task run.....");
    }
}
```

초 분 시간 일 월 요일 (연도)  
 \ | / / / /  
 0 \* \* \* \* \* (\*)

0/1 \* \* \* \* ? #1초 간격

0 0/1 \* \* \* ? #1분 간격

0 0 0/1 \* \* ? #1시간 간격

0 0 0 \* \* ? #매일 0시 마다

0 0 0 1 \* ? #매월 1일 마다

1,10,20 \* ? #매월 1일, 10일, 20일 마다

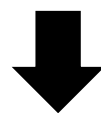
## 스케줄링 사용하기

## 지난 주문이력 삭제

쇼핑몰 주문정보의 통계를 구했으면, 지난 주문이력은 삭제한다.

오늘 날짜를 확인하고 전날 데이터를 삭제한다.

	123 no ▼	ABC product_name ▼	123 price ▼	reg_date ▼
1	1	무선 이어폰	75,000	2024-08-03 17:13:27.526
2	2	블루투스 스피커	50,000	2024-08-04 17:13:32.153
3	3	스마트 워치	120,000	2024-08-04 11:27:13.237
4	4	노트북	1,500,000	2024-08-05 11:27:15.482
5	5	게이밍 마우스	45,000	2024-08-05 11:27:17.423



	123 no ▼	ABC product_name ▼	123 price ▼	reg_date ▼
1	1	무선 이어폰	75,000	2024-08-03 17:13:27.526
2	2	블루투스 스피커	50,000	2024-08-04 17:13:32.153
3	3	스마트 워치	120,000	2024-08-04 11:27:13.237

```
Console × Problems Debug Shell
project13 - Project13Application [Spring Boot App] [pid: 196596]
2024-08-05 11:27:15.482 INFO 196596 --- [project13]
task run.....
OrderDTO(no=4, productName=노트북, price=1500000, regDate=2024-08-05 11:27:15.482)
4 remove..
OrderDTO(no=5, productName=게이밍 마우스, price=45000, regDate=2024-08-05 11:27:17.423)
5 remove..
```

# 스케줄링 사용하기

## 지난 주문이력 삭제

매일 1시가 되면 전날 들어온 주문 이력을 찾아서 삭제한다.

오늘 날짜에서 하루를 빼서 전날을 구한다.

```
// cron: 주기 (매일 1시가 될때마다 실행)
@Scheduled(cron = "0 0 1 * * * ")
public void removeOrderHistory() {
    System.out.println("task run.....");
    List<OrderDTO> list = orderService.getList();

    LocalDate now = LocalDate.now(); // 현재 날짜를 구함
    LocalDate yesterday = now.minusDays(1); // 전날을 구함

    // 전날 들어온 주문이력을 찾아서 삭제
    list.stream().forEach(dto -> {
        if (dto.getRegDate().toLocalDate().equals(yesterday)) {
            System.out.println(dto);
            orderService.remove(dto.getNo());
            System.out.println(dto.getNo() + " remove..");
        }
    });
}
```