

# 패스트 캠퍼스 포인트 만료하기

## 3 프로젝트 구현

# 프로젝트 생성하기

3.  
프로젝트 구현

## 패스트 캠퍼스 포인트 관리하기 프로젝트

git clone <https://github.com/kker5/point-management-practice>

패스트캠퍼스와는 아무런 관련이 없고 예시 프로젝트를 만든 것 입니다.

kker5 / point-management-practice Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

kker5 and kker5 Add point\_reservation 테이블 생성 ddl 추가 17d4f27 2 minutes ago 7 commits

gradle/wrapper	Initialize Project 초기화	2 days ago
src	Add point_reservation 테이블 생성 ddl 추가	2 minutes ago
.gitignore	Initialize Project 초기화	2 days ago
README.md	Initial commit	2 days ago
build.gradle	Initialize Project 초기화	2 days ago
gradlew	Initialize Project 초기화	2 days ago
gradlew.bat	Initialize Project 초기화	2 days ago

README.md

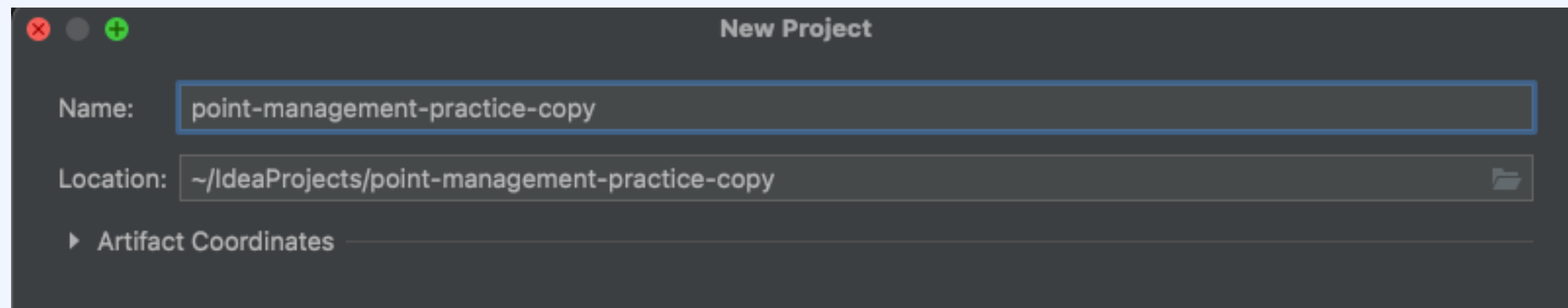
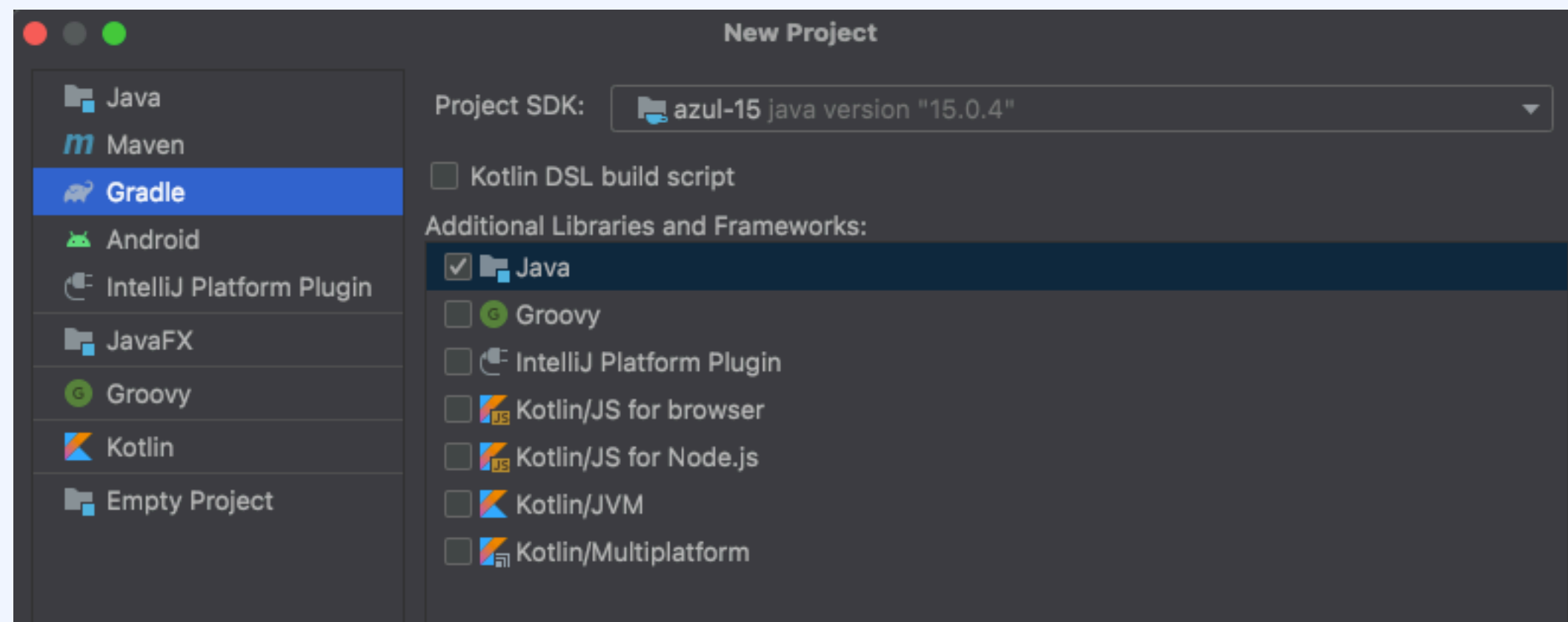
### point-management-practice

포인트 관리하기 예시 SpringBatch 프로젝트

# 프로젝트 생성하기

3.  
프로젝트 구현

## 프로젝트 생성하기



# Gradle로 의존성 설정하기

3.

프로젝트 구현

## Spring Batch

```
implementation 'org.springframework.boot:spring-boot-starter-batch'
```

## Spring Data JPA

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

## MySQL

```
implementation 'mysql:mysql-connector-java'
testRuntimeOnly 'com.h2database:h2'
```

## QueryDSL

```
// querydsl
implementation 'com.querydsl:querydsl-jpa'
implementation 'com.querydsl:querydsl-core'
annotationProcessor 'com.querydsl:querydsl-apt'
annotationProcessor group: 'com.querydsl', name: 'querydsl-apt', classifier: 'jpa'
annotationProcessor 'jakarta.persistence:jakarta.persistence-api'
annotationProcessor 'jakarta.annotation:jakarta.annotation-api'
```

# Gradle로 의존성 설정하기

3.

프로젝트 구현

## Lombok

```
compileOnly 'org.projectlombok:lombok'  
annotationProcessor 'org.projectlombok:lombok'
```

## Test

```
// asertj  
testImplementation 'org.assertj:assertj-core'  
// spring test  
testImplementation 'org.springframework.batch:spring-batch-test'  
testImplementation 'org.springframework.boot:spring-boot-starter-test'  
// junit  
testImplementation 'org.junit.jupiter:junit-jupiter-api'  
testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine'
```

# Gradle로 의존성 설정하기

3.

프로젝트 구현

## Plugin

```
plugins {  
    id 'java'  
    id 'org.springframework.boot' version '2.5.4'  
    id "io.spring.dependency-management" version "1.0.11.RELEASE"  
}
```

## repositories

```
repositories {  
    mavenCentral()  
}
```

## querydsl version

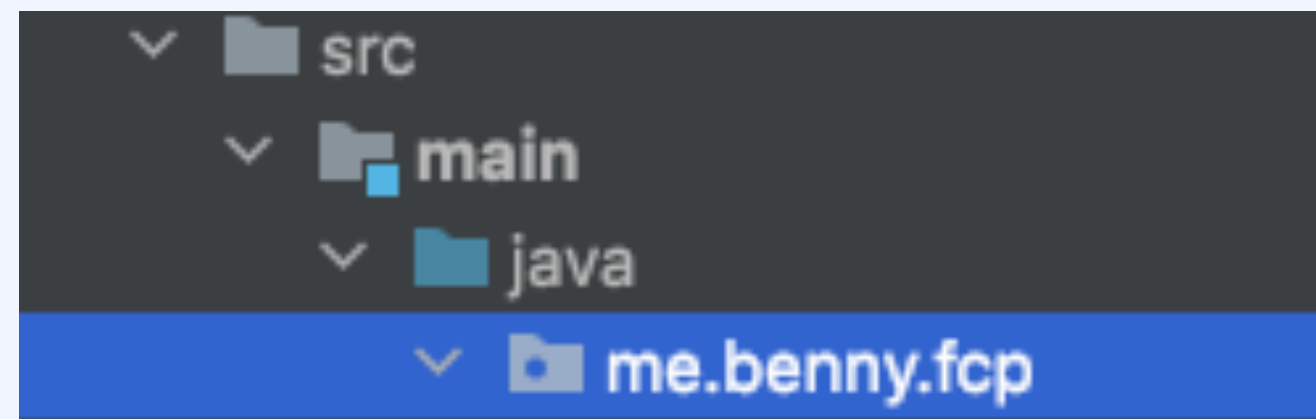
```
def querydslVersion = '5.0.0'
```

# Batch Application 만들기

3.

프로젝트 구현

## 패키지 생성



## main 함수 생성

```
@SpringBootApplication
@Slf4j
public class PointManagementApplication {
    public static void main(String[] args) {
        log.info("application arguments : " + String.join(",", args));
        SpringApplication.run(PointManagementApplication.class, args);
    }
}
```

# Batch Application 만들기

3.

프로젝트 구현

## BatchConfig

```
@EnableBatchProcessing
@Configuration
public class BatchConfig {
}
```

EnableBatchProcessing를 추가하면 사용할 수 있게 되는 Bean들

JobRepository (bean name "jobRepository")

JobLauncher (bean name "jobLauncher")

JobRegistry (bean name "jobRegistry")

JobExplorer (bean name "jobExplorer")

PlatformTransactionManager (bean name "transactionManager")

JobBuilderFactory (bean name "jobBuilders")

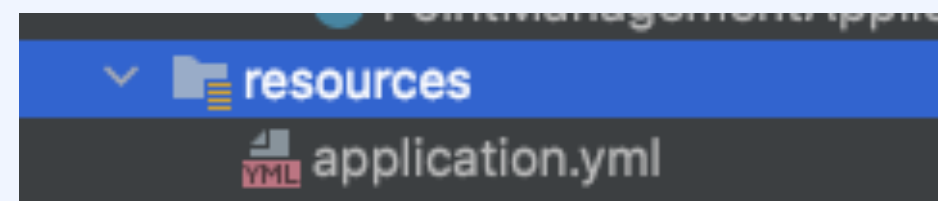
StepBuilderFactory (bean name "stepBuilders")



# Batch Application 만들기

## 3. 프로젝트 구현

### application.yml



```
spring:
  batch:
    job:
      names: ${job.name:NONE} # spring.batch.job.names를 job.name으로 치환
  jdbc:
    initialize-schema: always # batch에서 사용하는 스키마 생성여부를 always로 변경
  jpa:
    show-sql: true # sql 로그로 남기기를 true로 변경
    hibernate:
      ddl-auto: validate # entity를 보고 자동으로 데이터베이스 생성 여부를 validate (생성은 안하고 검증만)로 변경
```

# Batch Application 만들기

## spring.batch.job.names: \${job.name:NONE}

job이름을 넣어주면 그 Job이름을 보고 job을 실행시킵니다.

원래 대로라면 job 이름을 줄때 아래와 같이 줄수 있습니다.

```
java -jar batch.jar --spring.batch.job.names=expirePointJob
```

하지만 이 설정을 넣어주면 spring.batch.job.names를 job.name으로 치환해서 넣을 수 있게 됩니다.

원래는 Job 이름을 넣지 않고 프로젝트를 실행시키면 모든 Job이 실행됩니다.

그러나 기본 Job이름을 NONE으로 설정해두면 job이름을 주지 않고 프로젝트를 실행시키면 그냥 아무런 Job도 실행시키지 않습니다.

```
java -jar batch.jar --job.name=expirePointJob
```

## spring.batch.jdbc.initialize-schema: always

Spring Batch는 배치의 중간 상태나 결과를 데이터베이스에 남길 수 있습니다.

여러분들이 실행한 배치 기록을 남기겠다는 거죠.

따라서 데이터베이스에 기록 저장용 스키마와 테이블이 있어야합니다.

이 테이블을 서버가 뜰때 확인해서 없으면 자동으로 만들어 줄 것인지 물어보는 겁니다.

만약에 테이블이 없다면 자동으로 만들어 줄겁니다.

# Batch Application 만들기

## spring.jpa.show-sql : true

jpa가 Database에 요청하는 쿼리를 보여줄 것인지 아닌지 결정하는 값입니다.

true이면 쿼리를 보여줍니다.

다만, 쿼리가 너무 많이 찍히게 되면 로그파일의 크기가 커지고 로그를 보기 어려워질 수도 있어서 상황에 맞게 사용해야합니다.

## spring.jpa.hibernate.ddl-auto: validate

코드에서 선언한 Entity를 보고 실제 database에 DDL을 반영할지 선택하는 값입니다.

- none: 아무런 행동도 하지 않습니다.
- update: Database와 다른점이 있는지 확인하고 다른 부분만 변경합니다.
- validate: Database와 코드가 다른지 확인합니다. 다르면 에러가 발생되고 종료됩니다.
- create: 프로그램이 시작할 때 모든 Database를 드랍하고 코드를 보고 새로 만듭니다.
- create-drop: 프로그램이 시작할 때 모든 Database를 드랍하고 코드를 보고 새로 만듭니다. 그리고 프로그램이 종료되면 다시 Database를 드랍합니다.

# docker mysql 연동하기

3.

프로젝트 구현

## Docker로 MySQL을 설치한다고?

실제로 운영환경에서 데이터 베이스(MySQL)를 도커에 설치하는 경우는 없습니다.

그러나 많은 개발자들은 개발을 위해서 개인 PC환경에 MySQL를 설치하고 싶어하지 않습니다.

그 이유는 데이터 베이스를 설치하게되면 PC의 많은 리소스를 소모하게 되어 개인 PC의 성능저하를 일으킬 수 있기 때문입니다.

또한, 데이터베이스가 불필요해져서 설치 후 삭제를 할 때 귀찮은 과정이 많고 삭제가 정말 올바르게 되었는지 짹짹합니다.

그래서 도커를 설치하여 MySQL를 도커 컨테이너로 사용하고 불필요해지면 컨테이너만 삭제하면 되는 깔끔한 방식을 사용하겠습니다.

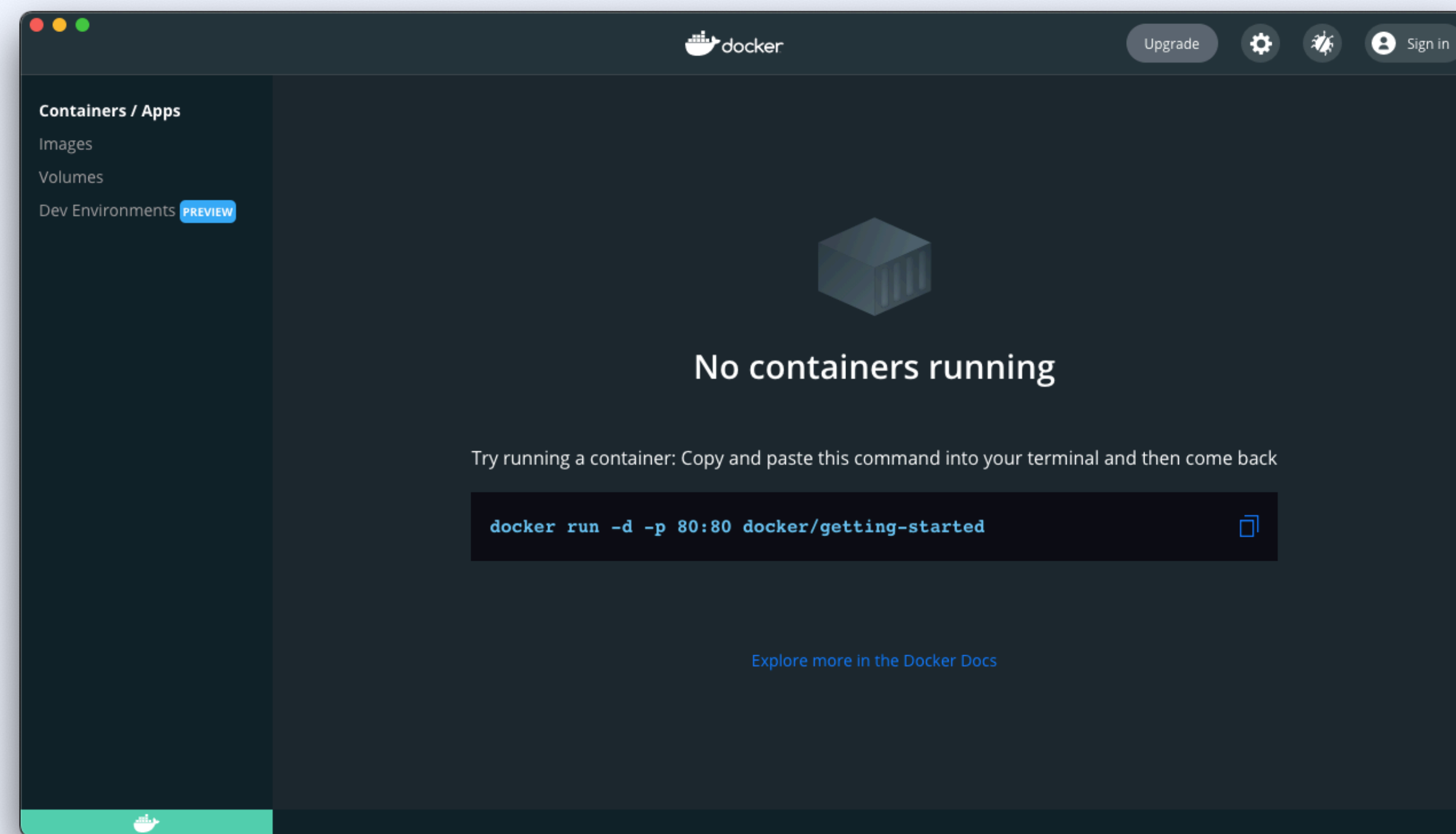
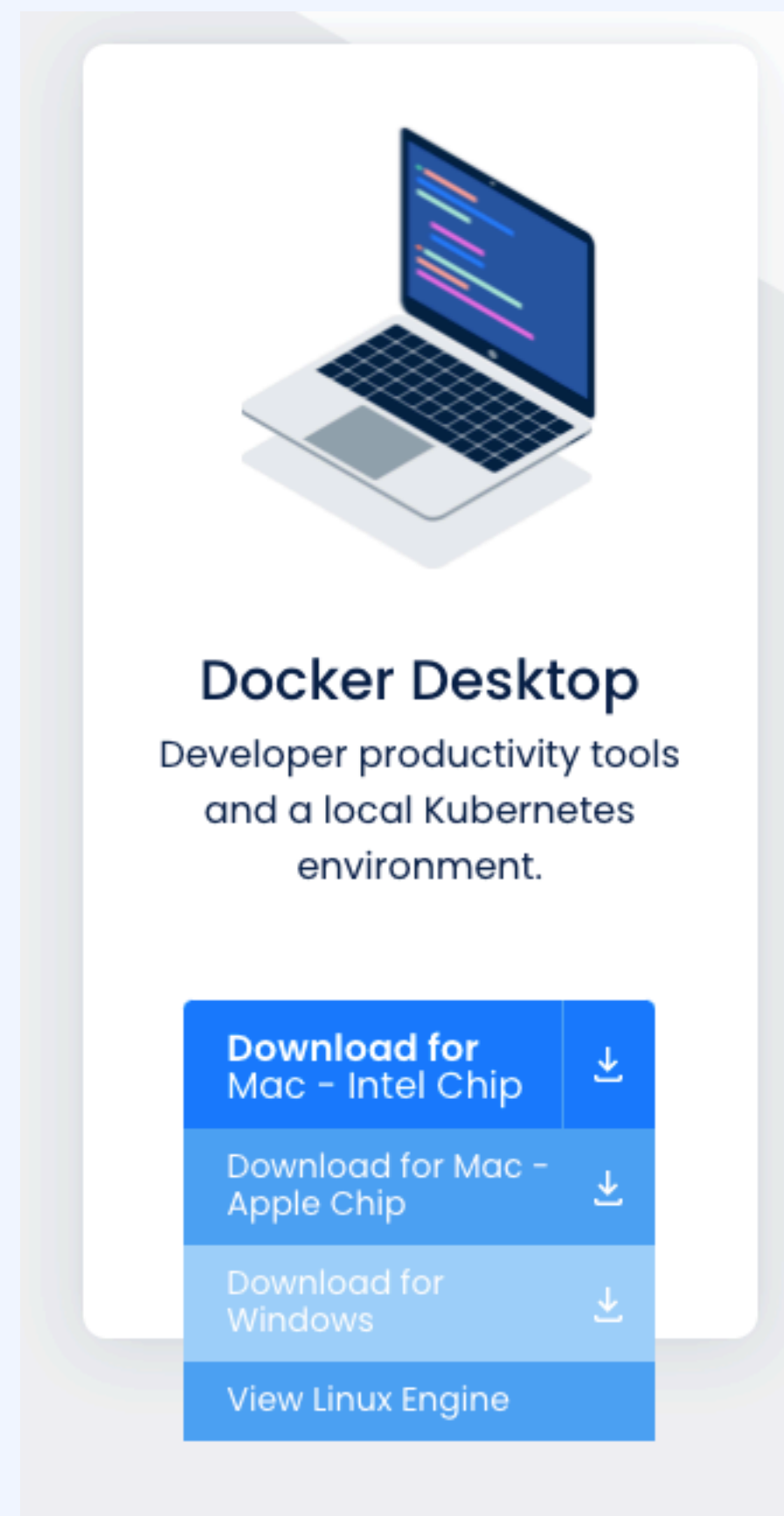
(개발환경을 위한 별도의 MySQL 서버가 준비되어있거나 개인PC에 그냥 설치해서 쓰시겠다고 해도 전혀 문제가 없습니다.)

# docker mysql 연동하기

3.  
프로젝트 구현

## Docker 설치

<https://www.docker.com/get-started>

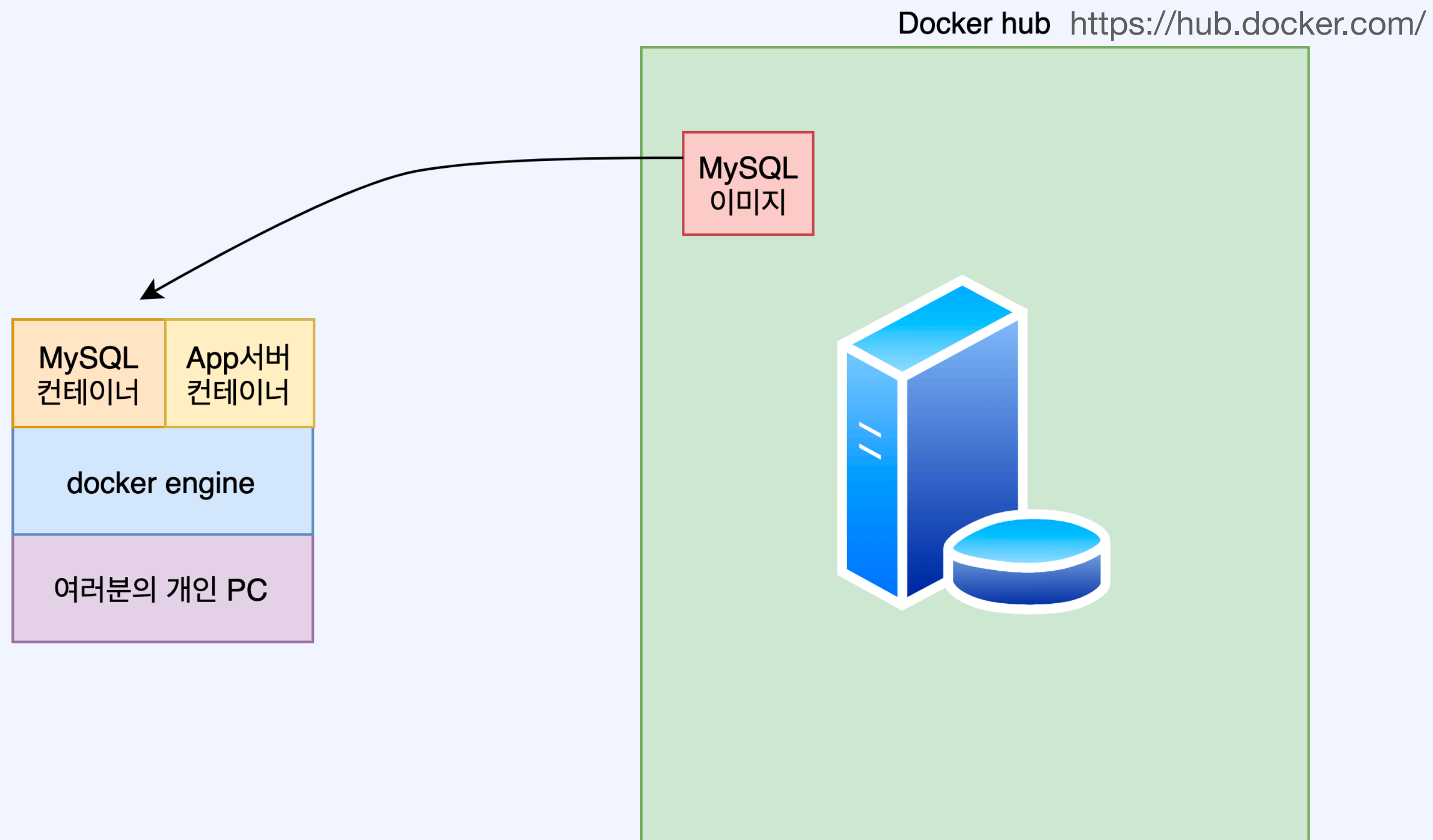


docker desktop

# docker mysql 연동하기

## 3. 프로젝트 구현

### Docker Image & Container



# docker mysql 연동하기

3.

프로젝트 구현

## Docker MySQL RUN

docker run : 도커 컨테이너 실행

-p 33060:3306 : 포트 파인딩 컨테이너 내부의 3306포트를 외부의 33060와 연결한다.

--name : 컨테이너 이름

-e : 컨테이너의 환경변수 지정 (MYSQL\_ROOT\_PASSWORD=password 을 통해 password를 password로 지정함)

-d : 컨테이너 실행은 백그라운드에서 진행

애플실리콘

```
docker run -p 33060:3306 --name point-mysql -e MYSQL_ROOT_PASSWORD=password --platform linux/amd64 -d mysql:8.0.26
```

그 외

```
docker run -p 33060:3306 --name point-mysql -e MYSQL_ROOT_PASSWORD=password -d mysql:8.0.26
```



# docker mysql 연동하기

3.

프로젝트 구현

## Docker MySQL 접속해보기

### 컨테이너 bash 실행

```
docker exec -i -t <container Id> /bin/bash
```

```
dasom@DASOMui-MacBookAir libs % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                                                                 NAMES
05c895a64cea   mysql:latest   "docker-entrypoint.s..." 6 minutes ago  Up 6 minutes  33060/tcp, 0.0.0.0:33060->3306/tcp, :::33060->3306/tcp  point-mysql
dasom@DASOMui-MacBookAir libs % docker exec -i -t 05c895a64cea /bin/bash
root@05c895a64cea:/#
```

mysql 을 실행시킵니다. (password를 물어보면 'password'를 입력합니다.)

```
mysql -u root -p
```

root의 password를 초기화해줍니다.

```
alter user 'root'@'%' identified with mysql_native_password by 'password';
```



# docker mysql 연동하기

3.  
프로젝트 구현

## Docker MySQL 컨테이너 삭제하기

MySQL를 삭제하려고 한다면 간단하게 MySQL 컨테이너를 삭제하면 됩니다.

### 컨테이너 조회

```
docker ps
```

```
dasom@DASOMui-MacBookAir point-management-practice % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                                                                 NAMES
83feba56c16e   mysql:8.0.26   "docker-entrypoint.s..." 9 seconds ago  Up 9 seconds  33060/tcp, 0.0.0.0:33060->3306/tcp, :::33060->3306/tcp  point-mysql
```

### 컨테이너 Stop

```
docker stop <container Id>
```

### 컨테이너 삭제

```
docker rm <container Id>
```

```
dasom@DASOMui-MacBookAir libs % docker stop 571ea9949060
571ea9949060
dasom@DASOMui-MacBookAir libs % docker rm 571ea9949060
571ea9949060
```

# docker mysql 연동하기

3.

프로젝트 구현

## Docker MySQL 이미지 삭제하기

MySQL를 삭제하려고 한다면 간단하게 MySQL 컨테이너를 삭제하면 됩니다.

### 이미지 가져오기

```
docker pull <image이름:버전>
```

### 이미지 조회

```
docker images
```

```
dasom@DASOMui-MacBookAir point-management-practice % docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
mysql         8.0.26    0716d6ebcc1a   6 days ago    514MB
```

### 이미지 삭제

```
docker rmi <image Id>
```

```
dasom@DASOMui-MacBookAir point-management-practice % docker rmi 0716d6ebcc1a
Untagged: mysql:8.0.26
Untagged: mysql@sha256:99e0989e7e3797cfbdb8d51a19d32c8d286dd8862794d01a547651a896bcf00c
Deleted: sha256:0716d6ebcc1a61c5a296fcb187e71f93531e510d4e4400267e2e502103d0194c
```

# MySQL Table 생성하기

3.

프로젝트 구현

## Database 생성

### MySQL 실행

```
docker exec -i -t <container Id> /bin/bash  
mysql -u root -p
```

### Database 생성

```
create database point;
```

### Database 사용

```
use point;
```

# MySQL Table 생성하기

## Table 생성

	컬럼 타입	타입 특징	필드 특징
id	bigint	매우 큰 숫자 표현 8byte	AUTO_INCREMENT Primary Key
amount	bigint	매우 큰 숫자 표현 8byte	
user_id	varchar(20)	char 타입과는 다르게 가변 문자열에 쓰임	
is_executed	tinyint	boolean 타입에 많이 씀 1byte	
expired_date	date	날짜를 표현할 때 사용	
content	text	매우 긴 text까지 지원	메시지 내용

# MySQL Table 생성하기

3.

프로젝트 구현

## Table 생성

### 포인트 지갑

```
CREATE TABLE `point_wallet` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `amount` bigint NOT NULL COMMENT '보유금액',
  `user_id` varchar(20) NOT NULL COMMENT '유저 ID',
  PRIMARY KEY (`id`),
) COMMENT '포인트지갑';
```

### 포인트 예약

```
CREATE TABLE `point_reservation` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `amount` bigint NOT NULL COMMENT '적립금액',
  `available_days` int NOT NULL COMMENT '유효기간',
  `earned_date` date NOT NULL COMMENT '적립일자',
  `is_executed` tinyint NOT NULL COMMENT '적용여부',
  `point_wallet_id` bigint NOT NULL COMMENT '포인트 지갑 ID',
  PRIMARY KEY (`id`),
) COMMENT '포인트 예약';
```

# MySQL Table 생성하기

3.

프로젝트 구현

## Table 생성

### Point 적립 내역

```
CREATE TABLE `point` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `amount` bigint NOT NULL COMMENT '적립금액',
  `earned_date` date NOT NULL COMMENT '적립일자',
  `expire_date` date NOT NULL COMMENT '만료일자',
  `is_used` tinyint NOT NULL COMMENT '사용유무',
  `is_expired` tinyint NOT NULL COMMENT '만료여부',
  `point_wallet_id` bigint NOT NULL COMMENT '포인트 지갑 ID',
  PRIMARY KEY (`id`)
) COMMENT '포인트적립내역';
```

### 메시지

```
CREATE TABLE `message` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `user_id` varchar(20) NOT NULL COMMENT '유저 ID',
  `title` varchar(200) NOT NULL COMMENT '제목',
  `content` text NOT NULL COMMENT '내용',
  PRIMARY KEY (`id`)
) COMMENT '메시지';
```

# MySQL Table 생성하기

## Application의 datasource 설정

```
//      runtimeOnly 'com.h2database:h2'
      implementation 'mysql:mysql-connector-java'
      testRuntimeOnly 'com.h2database:h2'
```

```
datasource: # datasource 정의
  url: jdbc:mysql://127.0.0.1:33060/point?useUnicode=true&characterEncoding=utf8
  driver-class-name: com.mysql.cj.jdbc.Driver # 드라이버 클래스명을 mysql로 지정
  username: root
  password: password
```

 docker run -p 33060:3306 --name point-mysql -e MYSQL\_ROOT\_PASSWORD=password -d mysql:8.0.26

# Entity 생성하기

3.

프로젝트 구현

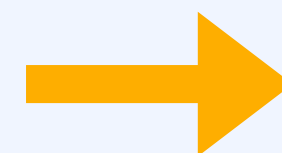
## IdEntity

모든 테이블에는 `id` bigint NOT NULL AUTO\_INCREMENT 가 포함되어 있습니다.  
따라서 아래와 같이 Id 생성 규칙을 일괄적으로 구현하고 모든 Entity가 IdEntity를 상속해서 Id를 따로 구현하지 않아도 되게 합니다.

```
@MappedSuperclass
@Getter
public abstract class IdEntity implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}
```

Id 생성을 Database에 위임합니다.  
즉, AUTO\_INCREMENT로 Id를 생성합니다.

```
public class PointWallet {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;
    @Column(name = "user_id", unique = true, nullable = false)
    String userId;
    @Column(name = "amount", columnDefinition = "BIGINT")
    BigInteger amount;
}
```



```
public class PointWallet extends IdEntity {
    @Column(name = "user_id", unique = true, nullable = false)
    String userId;
    @Column(name = "amount", columnDefinition = "BIGINT")
    BigInteger amount;
}
```



# Entity 생성하기

## 3. 프로젝트 구현

### PointWallet

포인트 지갑 Entity를 구현해보겠습니다.

```
CREATE TABLE `point_wallet` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `amount` bigint NOT NULL COMMENT '보유금액',
  `user_id` varchar(20) NOT NULL COMMENT '유저 ID',
  PRIMARY KEY (`id`),
) COMMENT '포인트지갑';
```

`@Entity` 이 클래스는 Entity 입니다.

`@Table` 이 클래스에 맞는 테이블을 가지고 있습니다. 그 이름은 point\_wallet입니다.

`@NoArgsConstructor` (access = AccessLevel.PROTECTED) protected인 기본 생성자를 자동으로 생성합니다.

`@AllArgsConstructor` 모든 인자를 다 받는 생성자를 public으로 자동으로 생성합니다.

`@Getter`

```
public class PointWallet extends IdEntity {
```

```
    // user 식별자
```

`@Column` (name = "user\_id", unique = true, nullable = false) user\_id라는 이름의 컬럼과 매핑합니다. unique하고 null이면 안됩니다.

```
    String userId; // 임의로 결정
```

```
    // 포인트 금액
```

`@Column` (name = "amount", columnDefinition = "BIGINT") amount라는 이름의 컬럼과 매핑합니다. 그 타입은 BIGINT입니다.

```
    BigInteger amount;
```

```
}
```

# Entity 생성하기

3.

프로젝트 구현

## Point

포인트 적립 Entity를 구현해보겠습니다.

```
public class Point extends IdEntity {
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "point_wallet_id", nullable = false)
    PointWallet pointWallet; // 포인트 지갑
    @Column(name = "amount", nullable = false, columnDefinition = "BIGINT")
    BigInteger amount; // 적립금액
    @Column(name = "earned_date", nullable = false)
    LocalDate earnedDate; // 적립일자
    @Column(name = "expire_date", nullable = false)
    LocalDate expireDate; // 만료일자
    @Column(name = "is_used", nullable = false, columnDefinition = "TINYINT", length = 1)
    boolean used; // 사용유무
    @Column(name = "is_expired", nullable = false, columnDefinition = "TINYINT", length = 1)
    boolean expired; // 만료유무
}
```

N(point) : 1(wallet)로 매핑됩니다. Lazy하게 가져옵니다.

point\_wallet\_id 컬럼으로 조인할 수 있습니다.

컬럼 타입이 TINYINT 입니다.

# Entity 생성하기

3.

프로젝트 구현

## PointReservation

포인트 예약 적립 Entity를 구현해보겠습니다.

```
public class PointReservation extends IdEntity {
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "point_wallet_id", nullable = false)
    PointWallet pointWallet; // 적립할 지갑

    @Column(name = "amount", nullable = false, columnDefinition = "BIGINT")
    BigInteger amount; // 적립금액

    @Column(name = "earned_date", nullable = false)
    LocalDate earnedDate; // 적립일자

    @Column(name = "available_days", nullable = false)
    int availableDays; // 유효일

    @Column(name = "is_executed", columnDefinition = "TINYINT", length = 1, nullable = false)
    boolean executed; // 실행여부

    public LocalDate getExpireDate() {
        return this.earnedDate.plusDays(this.availableDays);
    }
}
```

# Entity 생성하기

3.

프로젝트 구현

## Message

메시지 Entity를 구현해보겠습니다.

```
public class Message extends IdEntity {  
    // user 식별자  
    @Column(name = "user_id", unique = true, nullable = false)  
    String userId;  
    // 메시지 제목  
    @Column(name = "title", nullable = false)  
    String title;  
    // 메시지 내용  
    @Column(name = "content", nullable = false, columnDefinition = "text")  
    String content;  
    public static Message expiredPointMessageInstance() {  
    }  
    public static Message expireSoonPointMessageInstance() {  
    }  
}
```

# 테스트 구현하기

3.

프로젝트 구현

## 테스트 기초

Test는 given when then 절로 구분해서 구현합니다.

이런 구분을 명시적으로 해주는 테스트 툴도 있지만 대부분은 개발자가 자체적으로 구분해서 처리합니다.

예를 들면 코드에 주석으로 // given 이렇게 쓰고 그 아래에 given절을 구현합니다.

given : 테스트를 하기 위한 조건을 세팅합니다.

when : 테스트하고자 하는 메소드나 대상을 실행합니다.

then : 테스트 결과를 검증합니다.

```
@Test
void 더하기_테스트() {
    // given
    int firstNum = 1;
    int secondNum = 2;
    // when
    int result = firstNum + secondNum;
    //then
    assertEquals(3, result);
}
```

# 테스트 구현하기

## 3. 프로젝트 구현

### assertj

testImplementation 'org.assertj:assertj-core'

then 절의 검증 코드를 더 쉽게 구현해주는 라이브러리 입니다.  
사실상 여러분이 필요하다고 생각하는 모든 검증 메소드가 포함되어있습니다.

<code>then(3).isEqualTo(3);</code>	3 = 3
<code>then(List.of(1,2,3,4)).hasSize(4);</code>	리스트의 길이는 4
<code>then("abcdef").startsWith("abc");</code>	문자열은 abc로 시작
<code>then(BigInteger.valueOf(1000)).isEqualByComparingTo(BigInteger.valueOf(1000));</code>	BigInteger 1000은 BigInteger 1000
<code>then(LocalDate.of(2020, 5, 5)).isAfter(LocalDate.of(1999, 4, 3));</code>	2020-05-05 > 1999-04-03

```
@Test
void 더하기_테스트() {
    // given
    int firstNum = 1;
    int secondNum = 2;
    // when
    int result = firstNum + secondNum;
    //then
    assertEquals(3, result);
}
```

→

```
then(result).isEqualTo(3);
```

# 테스트 구현하기

3.

프로젝트 구현

## BatchTestSupport

단위테스트가 아닌 Batch Test를 할 때는 여러가지 설정을 해줘야합니다.

```
@SpringBootTest 스프링의 통합 테스트를 제공
@ActiveProfiles("test") 프로파일(구동환경)이 test
public abstract class BatchTestSupport {
    protected JobExecution launchJob(Job job, JobParameters jobParameters) throws Exception {
    }
    @AfterEach 매번 테스트가 끝난 뒤에 할 일 명시합니다.
    protected void deleteAll() { Job테스트는 자동으로 rollback되지 않아서 임의로 데이터를 삭제해줘야합니다.
    }
}

class SampleJobConfigurationTest extends BatchTestSupport {
    @Autowired
    Job sampleJob;
    @Test
    void sampleJob_success() throws Exception {
        JobExecution execution = launchJob(sampleJob, jobParameters);
        then(execution.getExitStatus()).isEqualTo(ExitStatus.COMPLETED);
    }
}
```

Job을 실행시킵니다.

Job의 실행 결과를 확인합니다.

# 테스트 구현하기

3.

프로젝트 구현

## 테스트 application.yml 설정

```
spring:
  batch:
    job:
      names: ${job.name:NONE}
  jdbc:
    initialize-schema: always
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: create-drop
```



# 포인트 만료하기

3.

프로젝트 구현

## Job 테스트 구현하기

Job name : expirePointJob

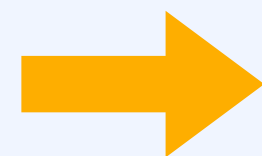
Job Parameter : today (오늘 일자)

아래에 경우에 해당하면 포인트 만료

1. 사용기한이 오늘보다 작음
2. 포인트의 상태가 만료가 아님
3. 포인트는 사용하지 않음

포인트 만료하는 방법

1. 포인트의 만료 여부를 true로 변경
2. 포인트 지갑의 금액을 차감



given

1. 사용기한이 오늘보다 작음
2. 포인트의 상태가 만료가 아님
3. 포인트는 사용하지 않음

when

expirePointJob 실행

then

1. 포인트의 만료 여부가 true인지 확인
2. 포인트 지갑의 금액을 차감되었는지 확인

# 포인트 만료하기

3.

프로젝트 구현

## Job 구현하기

Job name : expirePointJob

Job Parameter : today (오늘 일자)

```
@Bean
public Job expirePointJob(
    JobBuilderFactory jobBuilderFactory,
    TodayJobParameterValidator validator,
    Step expirePointStep
) {
    return jobBuilderFactory
        .get("expirePointJob")
        .validator(validator)
        .incrementer(new RunIdIncrementer())
        .start(expirePointStep)
        .build();
}
```

JobBuilderFactory 가져오기

Job Parameter에 today가 있는지 확인하는 validator

Step 가져오기

Job의 이름

같은 Job을 같은 JobParameter로 돌려도 해당 Id가 계속해서 증가하여  
job이 중복실행된 것으로 인지 하지 않음

# 포인트 만료하기

3.

프로젝트 구현

## Step 구현하기

```

@Bean
@JobScope Job에서 스텝을 실행할 때 아래의 스텝을 Lazy하게 생성함
public Step expirePointStep(
    StepBuilderFactory stepBuilderFactory, StepBuilderFactory 가져오기
    PlatformTransactionManager platformTransactionManager, TransactionManager 가져오기
    JpaPagingItemReader<Point> expirePointItemReader, ItemReader 가져오기
    ItemProcessor<Point, Point> expirePointItemProcessor, ItemProcessor 가져오기
    ItemWriter<Point> expirePointItemWriter ItemWriter 가져오기
) {
    return stepBuilderFactory
        .get("expirePointStep") Step 이름
        .allowStartIfComplete(true) Step 중복 실행 가능
        .transactionManager(platformTransactionManager)
        .<Point, Point>chunk(1000) Chunk Size는 1000
        .reader(expirePointItemReader)
        .processor(expirePointItemProcessor)
        .writer(expirePointItemWriter)
        .build();
}

```

# 포인트 만료하기

## 3. 프로젝트 구현

### Reader 구현하기

```

@Bean
@StepScope Step이 아래의 ItemReader를 Lazy하게 생성함
public JpaPagingItemReader<Point> expirePointItemReader(
    EntityManagerFactory entityManagerFactory,
    @Value("#{T(java.time.LocalDate).parse(jobParameters[today])}") JobParameter를 가져와서 LocalDate로 converting함
    LocalDate today
) {
    return new JpaPagingItemReaderBuilder<Point>() JpaPagingItemReader 생성
        .name("expirePointItemReader")
        .entityManagerFactory(entityManagerFactory) JPQL로 쿼리 작성
        .queryString("select p from Point p where p.expireDate < :today and used = false and expired = false")
        .parameterValues(Map.of("today", today)) :today에 파라미터값을 넣어줌
        .pageSize(1000) page 1개의 사이즈
        .build();
}

```

# 포인트 만료하기

## 3. 프로젝트 구현

### Processor 구현하기

```
@Bean
@StepScope
public ItemProcessor<Point, Point> expirePointItemProcessor() {
    return point -> {
        point.setExpired(true);
        PointWallet wallet = point.getPointWallet();
        wallet.setAmount(wallet.getAmount().subtract(point.getAmount()));
        return point;
    };
}
```

Point를 받아와서 수정하고 Point를 반환함

Point의 만료상태를 true로 바꿈

Point 안에 있는 PointWallet의 잔액을 차감함

# 포인트 만료하기

3.

프로젝트 구현

## Writer 구현하기

```
@Bean
@StepScope
public ItemWriter<Point> expirePointItemWriter(
    PointRepository pointRepository,
    PointWalletRepository pointWalletRepository
) {
    return points -> {
        for (Point point : points) {
            if (point.isExpired()) {
                pointRepository.save(point);
                pointWalletRepository.save(point.getPointWallet());
            }
        }
    };
}
```

Processor에서 수정한 Point와 PointWallet을 수정함

# 포인트 만료하기

3.

프로젝트 구현

## Validator 구현하기

```
@Component
public class TodayJobParameterValidator implements JobParametersValidator {
    @Override
    public void validate(JobParameters parameters) throws JobParametersInvalidException {
        if (parameters == null)
            throw new JobParametersInvalidException("job parameter today is required");
        String todayStr = parameters.getString("today");
        if (todayStr == null)
            throw new JobParametersInvalidException("job parameter today is required");
        try {
            LocalDate.parse(todayStr);
        } catch (DateTimeParseException ex) {
            throw new JobParametersInvalidException("job parameter today format is not valid");
        }
    }
}
```

JobParameter를 검증하는 Validator

today가 없으면 에러

today를 파싱하다 에러나면 에러

# 예약 포인트 적립하기

3.

프로젝트 구현

## Job 구현하기

Job name : executePointReservationJob

Job Parameter : today (오늘 일자)

```
@Bean
public Job executePointReservationJob(
    JobBuilderFactory jobBuilderFactory,
    TodayJobParameterValidator validator,
    Step executePointReservationStep
) {
    return jobBuilderFactory
        .get("executePointReservationJob")
        .validator(validator)
        .incrementer(new RunIdIncrementer())
        .start(executePointReservationStep)
        .build();
}
```



# 만료된 포인트 메시지 만들기

3.

프로젝트 구현

## Job 구현하기

Job name : messageExpiredPointJob

Job Parameter : today (오늘 일자)

예상 결과 : 메시지 저장

메시지 내용

- 제목 : 1000 포인트 만료
- 내용 : 2015-06-02 기준 1000 포인트가 만료되었습니다.

```
@Configuration
public class MessageExpiredPointJobConfiguration {
    @Bean
    public Job messageExpiredPointJob(
        JobBuilderFactory jobBuilderFactory,
        TodayJobParameterValidator validator,
        Step messageExpiredPointStep
    ) {
        return jobBuilderFactory
            .get("messageExpiredPointJob")
            .validator(validator)
            .incrementer(new RunIdIncrementer())
            .start(messageExpiredPointStep)
            .build();
    }
}
```

# 만료된 포인트 메시지 만들기

## 3. 프로젝트 구현

### Step 구현하기

```
@Bean
@JobScope
public Step messageExpiredPointStep(
    InputExpiredPointAlarmCriteriaDateStepListener listener, today로 알림 기준일을 구하는 listener
) {
    return stepBuilderFactory
        .get("messageExpiredPointStep")
        .allowStartIfComplete(true)
        .transactionManager(platformTransactionManager)
        .listener(listener)
        .<ExpiredPointSummary, Message>chunk(1000)
        .reader(messageExpiredPointItemReader)
        .processor(messageExpiredPointItemProcessor)
        .writer(messageExpiredPointItemWriter)
        .build();
}
```

# 만료된 포인트 메시지 만들기

3.  
프로젝트 구현

## StepExecutionListener 구현하기

StepExecutionListener 구현

```
@Component
public class InputExpiredPointAlarmCriteriaDateStepListener implements StepExecutionListener {
    @Override
    public void beforeStep(StepExecution stepExecution) {
        JobParameter todayParameter = stepExecution.getJobParameters().getParameters().get("today");
        if (todayParameter == null) {
            return; 가져온 값이 null이면 종료
        }
        LocalDate today = LocalDate.parse((String) todayParameter.getValue()); 가져온 값을 LocalDate로 변환
        ExecutionContext context = stepExecution.getExecutionContext();
        context.put("alarmCriteriaDate", today.minusDays(1).format(DateTimeFormatter.ISO_DATE));
        stepExecution.setExecutionContext(context); StepExecutionContext에 등록 알림 기준일자 = today -1
    }
}
```

# 만료된 포인트 메시지 만들기

## 3. 프로젝트 구현

### Reader 구현하기

```

@Bean
@StepScope
public RepositoryItemReader<ExpiredPointSummary> messageExpiredPointItemReader(
    PointRepository pointRepository,
    @Value("#{T(java.time.LocalDate).parse(stepExecutionContext[alarmCriteriaDate])}")
    LocalDate alarmCriteriaDate
) {
    return new RepositoryItemReaderBuilder<ExpiredPointSummary>()
        .name("messageExpiredPointItemReader")
        .repository(pointRepository)
        .methodName("sumByExpiredDate")
        .pageSize(1000)
        .arguments(alarmCriteriaDate)
        .sorts(Map.of("pointWallet", Sort.Direction.ASC))
        .build();
}

```

RepositoryItemReader

pointRepository.sumByExpiredDate

StepExecutionContext에서 알림 기준일을 가져옴

sort 기준이 ASC

## 만료된 포인트 메시지 만들기

### QueryDSL

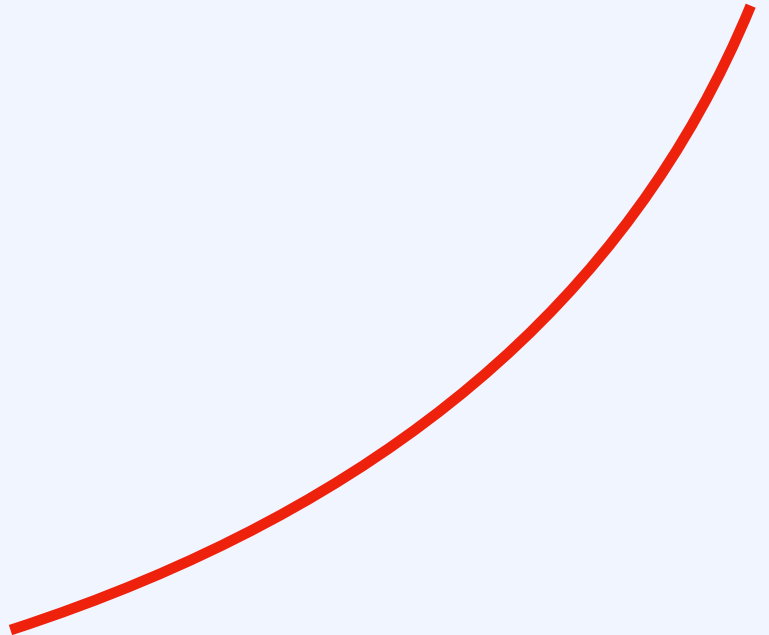
ItemReader에서 사용한 sumByExpiredDate 는 단순한 쿼리로 구현할 수 없습니다.

sum, group by, limit, order by, inner join 등 다양한 구문을 써야만 조회할 수 있습니다.

아래와 같이 문자열로 쿼리를 작성하는 건 실수를 유발할 수 있고 컴파일 시점에 문제가 파악되지 않고 보기에도 좋지 않습니다.

```
Page<ExpiredPointSummary> sumByExpiredDate(LocalDate alarmCriteriaDate, Pageable pageable);
```

```
select
w.user_id,
sum(p.amount)
from point p
inner join point_wallet w
on p.point_wallet_id = w.id
where p.is_expired = 1
and p.is_used = 0
and p.expire_date = '2021-01-01'
group by p.point_wallet_id
order by p.point_wallet_id asc
limit 1000, 0;
```



# 만료된 포인트 메시지 만들기

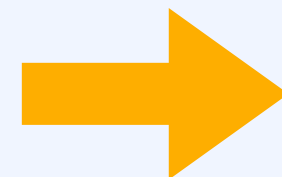
3.

프로젝트 구현

## QueryDSL

QueryDSL은 복잡한 쿼리를 DSL 형식의 자바 코드로 구현할 수 있도록 해주는 라이브러리입니다.

```
select
w.user_id,
sum(p.amount)
from point p
inner join point_wallet w
on p.point_wallet_id = w.id
where p.is_expired = 1
and p.is_used = 0
and p.expire_date = '2021-01-01'
group by p.point_wallet_id;
```



```
from(QPoint.point) from point
.select( 조회할 대상을 정의합니다.
        new QExpiredPointSummary(
            point.pointWallet.userId,
            point.amount.sum().coalesce(BigInteger.ZERO)
        )
        )
.where(point.expired.eq(true)) where 절에 is_expired = 1을 추가합니다.
.where(point.used.eq(false))
.where(point.expireDate.eq(alarmCriteriaDate))
.groupBy(point.pointWallet); group by합니다.
```

userId와 sum(amount)를 조회해서  
ExpiredPointSummary에 넣습니다.  
즉, 이 코드의 결과는 ExpiredPointSummary입니다.

# 만료된 포인트 메시지 만들기

3.

프로젝트 구현

## QueryDSL

QueryDSL을 사용하기 위해서는  
build.gradle의 의존성을 아래처럼 설정합니다.

```
// querydsl
implementation 'com.querydsl:querydsl-jpa'
implementation 'com.querydsl:querydsl-core'
annotationProcessor 'com.querydsl:querydsl-apt'
annotationProcessor group: 'com.querydsl', name: 'querydsl-apt', classifier: 'jpa'
annotationProcessor 'jakarta.persistence:jakarta.persistence-api'
annotationProcessor 'jakarta.annotation:jakarta.annotation-api'
```

annotationProcessor 덕분에 컴파일 하는 시점에 QClass를 만듭니다.



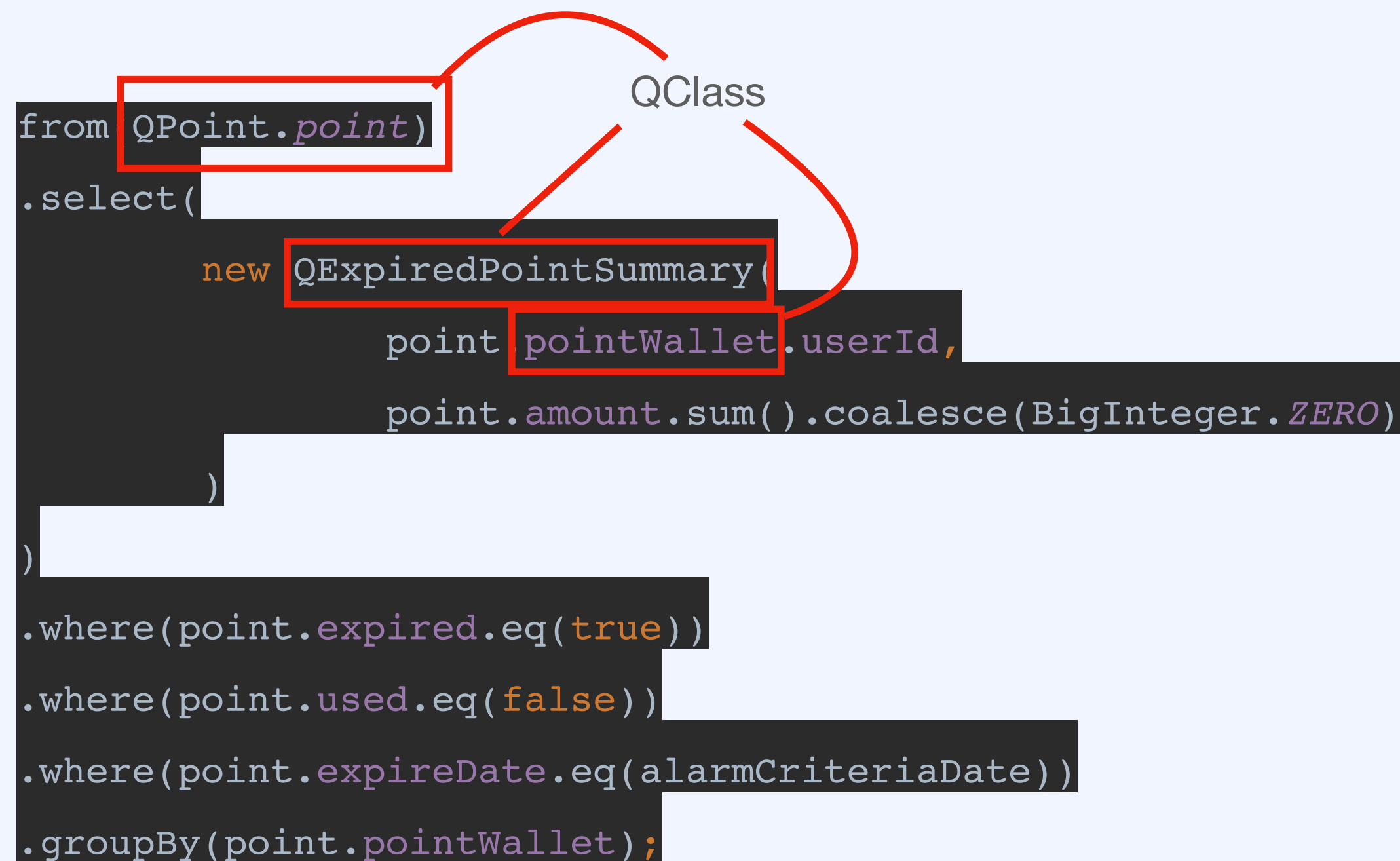
# 만료된 포인트 메시지 만들기

3.

프로젝트 구현

## QueryDSL

QueryDSL을 쓰기 위해서는 QClass를 만들어야합니다.  
QClass는 아래와 같이 Q + Entity 클래스의 이름 이렇게 생성되며  
컴파일 하는 시점에 자동으로 생성됩니다.



```
from QPoint.point)
.select(
    new QExpiredPointSummary(
        point.pointWallet.userId,
        point.amount.sum().coalesce(BigInteger.ZERO)
    )
)
.where(point.expired.eq(true))
.where(point.used.eq(false))
.where(point.expireDate.eq(alarmCriteriaDate))
.groupBy(point.pointWallet);
```

```
/**
 * QPoint is a Querydsl query type for Point
 */
@Generated("com.querydsl.codegen.EntitySerializer")
public class QPoint extends EntityPathBase<Point> {

    public static final QPoint point = new QPoint("point");
    public final NumberPath<java.math.BigInteger> amount = ...;
    public final DatePath<java.time.LocalDate> earnedDate = ...;
    public final BooleanPath expired = createBoolean("expired");
    public final DatePath<java.time.LocalDate> expireDate = ...;
    //inherited
    public final NumberPath<Long> id = _super.id;
    public final me.benny.fcp.point.wallet.QPointWallet pointWallet;
    public final BooleanPath used = createBoolean("used");
    public QPoint(Path<? extends Point> path) {
        ...
    }
}
```



# 만료된 포인트 메시지 만들기

3.  
프로젝트 구현

## QueryDSL로 PointRepository쿼리 작성하기

PointCustomRepository

```
public interface PointCustomRepository {
    Page<ExpiredPointSummary> sumByExpiredDate(LocalDate alarmCriteriaDate, Pageable pageable);
}
```

PointCustomRepositoryImpl

```
public class PointCustomRepositoryImpl extends QuerydslRepositorySupport implements PointCustomRepository {
    public PointCustomRepositoryImpl() {
        super(Point.class);
    }
}
```

# 만료된 포인트 메시지 만들기

3.

프로젝트 구현

## QueryDSL로 PointRepository 쿼리 작성하기

```
@Override
public Page<ExpiredPointSummary> sumByExpiredDate(LocalDate alarmCriteriaDate, Pageable pageable) {
    QPoint point = QPoint.point;
    JPQLQuery<ExpiredPointSummary> query =
        from(QPoint.point)
        .select(
            new QExpiredPointSummary(
                point.pointWallet.userId,
                point.amount.sum().coalesce(BigInteger.ZERO)
            )
        )
        .where(point.expired.eq(true))
        .where(point.used.eq(false))
        .where(point.expireDate.eq(alarmCriteriaDate))
        .groupBy(point.pointWallet);
}
```

# 만료된 포인트 메시지 만들기

## 3. 프로젝트 구현

### QueryDSL로 PointRepository 쿼리 작성하기

```
List<ExpiredPointSummary> expiredPointList = getQuerydsl().applyPagination(pageable, query).fetch();
long elementCount = query.fetchCount();
return new PageImpl<>(
    expiredPointList,
    PageRequest.of(pageable.getPageNumber(), pageable.getPageSize()),
    elementCount
);
```

QuerydslRepositorySupport에서 제공하는 Paging처리

전체 개수를 구하기 위해 Paging없이 count 개수를 구함

Page 정보 제공  
page  
size  
total element 개수  
total page 개수  
등등.. 제공

# 만료된 포인트 메시지 만들기

3.  
프로젝트 구현

## Processor 구현하기

```
@Bean
@StepScope
public ItemProcessor<ExpiredPointSummary, Message> messageExpiredPointItemProcessor(
    @Value("#{T(java.time.LocalDate).parse(jobParameters[today])}")
    LocalDate today
) {
    return summary -> Message.expiredPointMessageInstance(
        summary.getUserId(), today, summary.getAmount()
    );
}

public static Message expiredPointMessageInstance(
    String userId, LocalDate expiredDate, BigInteger expiredAmount
) {
    return new Message(
        userId,
        String.format("%s 포인트 만료", expiredAmount.toString()),
        String.format("%s 기준 %s 포인트가 만료되었습니다.", expiredDate.format(DateTimeFormatter.ISO_DATE), expiredAmount)
    );
}
```

## 만료된 포인트 메시지 만들기

3.

프로젝트 구현

### Writer 구현하기

```
@Bean
@StepScope
public JpaItemWriter<Message> messageExpiredPointItemWriter(
    EntityManagerFactory entityManagerFactory
) {
    JpaItemWriter<Message> jpaItemWriter = new JpaItemWriter<>();
    jpaItemWriter.setEntityManagerFactory(entityManagerFactory);
    return jpaItemWriter;
}
```

# 프로젝트 빌드, 실행시키기

3.

프로젝트 구현

## Intellij에서 실행하기

Program Arguments

Job 이름 넣기

--job.names=expirePointJob

Job Parameter 넣기

today=2021-09-01

```

11  @SpringBootApplication
12  @Slf4j
13  public class PointManagementApplication {
14      public static void main(String[] args) {
15          // Run 'PointManagemen....main()' ^C^R
16          // Debug 'PointManagemen....main()' ^C^D
17          // Run 'PointManagemen....main()' with Coverage
18          // Modify Run Configuration...
19      }
20  }

```

Name: 
☐ Store as project file

Build and run Modify options

CLI arguments to your application.

Working directory:

☐ Open run/debug tool window when started

Code Coverage Modify

# 프로젝트 빌드, 실행시키기

## 3. 프로젝트 구현

### IntelliJ에서 실행하기

결과 확인

Job 실행

Step 실행

JobParameter

```
Job: [SimpleJob: [name=expirePointJob]] launched with the following parameters: [{run.id=6, today=2021-09-03}]
org.springframework.batch.item.ItemWriter is an interface. The implementing class will not be queried for annotation
org.springframework.batch.item.ItemProcessor is an interface. The implementing class will not be queried for annotation
Executing step: [expirePointStep]
d3_1_, point0_.expire_date as expire_d4_1_, point0_.is_expired as is_expir5_1_, point0_.point_wallet_id as point_w
Step: [expirePointStep] executed in 141ms
Job: [SimpleJob: [name=expirePointJob]] completed with the following parameters: [{run.id=6, today=2021-09-03}] a
Closing JPA EntityManagerFactory for persistence unit 'default'
HikariPool-1 - Shutdown initiated...
HikariPool-1 - Shutdown completed.
```

Step 정상종료

Job 정상 종료



# 프로젝트 빌드, 실행시키기

3.

프로젝트 구현

## Jar로 만들어서 실행시키기

빌드를 해서 Jar생성  
위치 : build/libs

해당 Jar를 실행시키기

java -jar point-management-practice-copy-1.0-SNAPSHOT.jar --job.name=expirePointJob today=2021-09-03

```
dasom@DASOMui-MacBookAir point-management-practice-copy % ./gradlew clean build

> Task :test
2021-09-12 23:21:49.052 INFO 96154 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory
2021-09-12 23:21:49.052 INFO 96154 --- [ionShutdownHook] .SchemaDropperImpl$DelayedDropActionImpl : HHH000477: Starting delayed evict
ctory shut-down'
Hibernate: drop table if exists message CASCADE
Hibernate: drop table if exists point CASCADE
Hibernate: drop table if exists point_reservation CASCADE
Hibernate: drop table if exists point_wallet CASCADE
2021-09-12 23:21:49.055 INFO 96154 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated
2021-09-12 23:21:49.056 INFO 96154 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed

BUILD SUCCESSFUL in 10s
9 actionable tasks: 9 executed
dasom@DASOMui-MacBookAir point-management-practice-copy % cd build/libs
dasom@DASOMui-MacBookAir libs % ls
point-management-practice-copy-1.0-SNAPSHOT-plain.jar point-management-practice-copy-1.0-SNAPSHOT.jar
dasom@DASOMui-MacBookAir libs % java -jar point-management-practice-copy-1.0-SNAPSHOT.jar --job.name=expirePointJob today=2021-09-03
```

빌드 실행

jar 위치

Job 실행