Spring Boot Seminar - 3

Wafflestudio Rookies seminar 2024

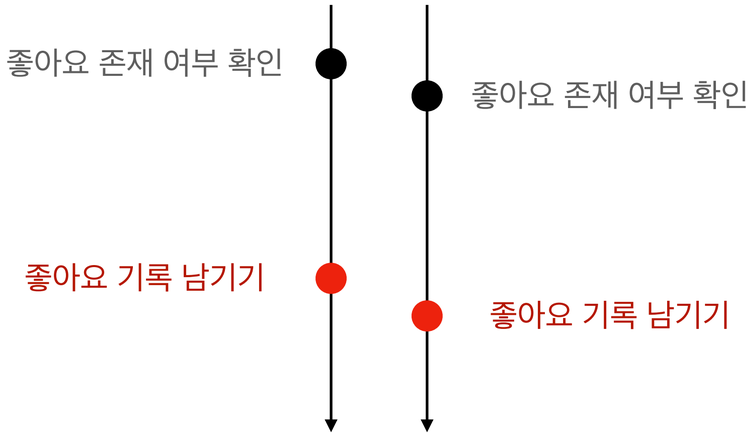
Sangmin Km, 2024

Seminar - 3 목차

- Assignment 2 리뷰
- Spring 뜯어보기
- Gradle 뜯어보기

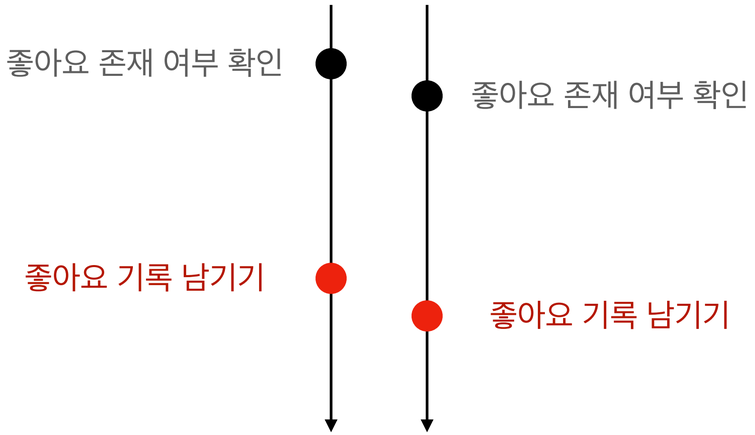
Assignment 2 리뷰

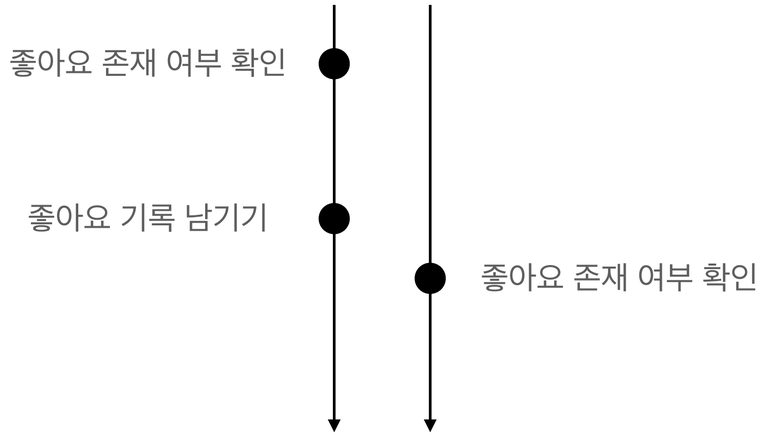
- ReviewLike 의 동시성 문제 해결
- 예상되는 문제 상황 - 한 사람의 좋아요 요청이 동시에 많이 들어오는 경우



Assignment 2 리뷰

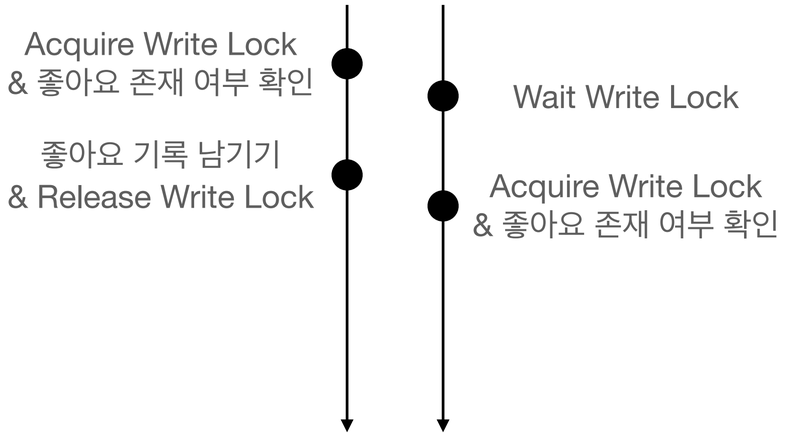
- ReviewLike 의 동시성 문제 해결
- Pessimistic Write Lock 으로 해결하기





Assignment 2 리뷰

- ReviewLike 의 동시성 문제 해결
- Pessimistic Write Lock 으로 해결하기



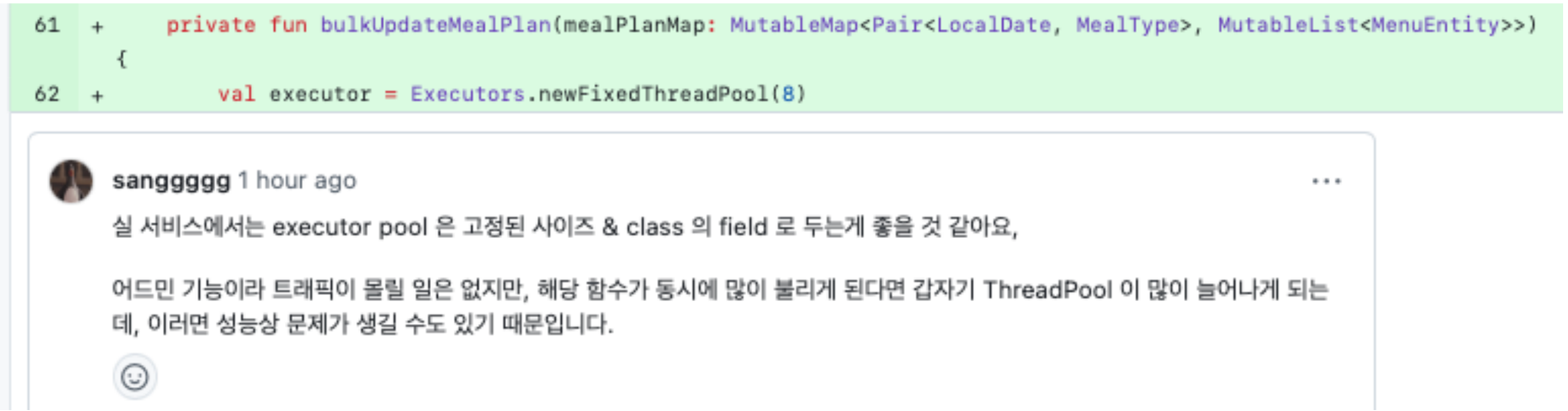

```kotlin
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("SELECT l FROM review_likes l WHERE l.review = :review AND l.author = :author")
fun findByReviewAndAuthor(
    review: ReviewEntity,
    author: UserEntity,
): ReviewLikeEntity?
```

Assignment 2 리뷰

- ReviewLike 의 동시성 문제 해결
- Unique Constraint 로 해결하기

```
@Entity(name = "review_likes")
@Table(
    name = "review_likes",
    indexes = [
        Index(name = "idx_review_id_user_id", columnList = "review_id, user_id", unique = true),
```

Assignment 2 리뷰



Spring 뜯어보기

Spring Core 가 하는 일

- Inversion of control (IoC)
  - Dependency Injection 을 통해 코드간의 결합도를 낮추어 준다.
- Aspect-Oriented Programming (AOP)
  - Cross-cutting 문제들을 비즈니스 로직과 분리하는 것을 쉽게 한다.
  - e.g. Transaction, Logging 등

Spring 뜯어보기

Inversion of Control

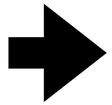
```java
public class OrderService {
    private CreditCardPayment paymentService;

    public OrderService() {
        this.paymentService = new CreditCardPayment(); // Direct dependency creation
    }

    public void placeOrder() {
        paymentService.processPayment();
        System.out.println("Order placed successfully.");
    }
}

public class CreditCardPayment {
    public void processPayment() {
        System.out.println("Processing payment with credit card.");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        OrderService orderService = new OrderService();
        orderService.placeOrder();
    }
}
```



```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class OrderService {
    private final PaymentService paymentService;

    @Autowired // Constructor-based dependency injection
    public OrderService(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    public void placeOrder() {
        paymentService.processPayment();
        System.out.println("Order placed successfully.");
    }
}
```

Spring 뜯어보기
Inversion of Control

- 하는 이유
  - 모듈화
  - Unit 테스트 용이
  - Configuration 에 대한 유연성
- 사실 셋 다 비슷한 말…

Spring 뜯어보기

Inversion of Control

- Configuration 에 대한 유연성
  - 서버 환경을 다양하게 구성하고 싶다
  - 테스트 / 실서버 분리
  - 결제를 테스트 서버에서 마구잡이로 하면…
- Spring Profile 기능을 통한 Configuration 가능
  - 내가 실행하고자 하는 환경을 config file 에 기록하기
  - `application.yaml`, `application.properties`

Step 1: Define the PaymentService Interface

kotlin

```kotlin
interface PaymentService {
    fun processPayment()
}
```

Step 2: Create Multiple Implementations of PaymentService

1. **Development/Testing Implementation** (MockPaymentService):

kotlin

```kotlin
import org.springframework.context.annotation.Profile
import org.springframework.stereotype.Service

@Service
@Profile("dev", "test") // Only active in 'dev' and 'test' profiles
class MockPaymentService : PaymentService {
    override fun processPayment() {
        println("Processing payment in mock environment.")
    }
}
```

2. **Production Implementation** (RealPaymentService):

kotlin

```kotlin
import org.springframework.context.annotation.Profile
import org.springframework.stereotype.Service

@Service
@Profile("prod") // Only active in the 'prod' profile
class RealPaymentService : PaymentService {
    override fun processPayment() {
        println("Processing payment in real production environment.")
    }
}
```

Spring 뜯어보기

Inversion of Control

- Spring Profile 기능을 통한 Configuration 가능
  - 내가 실행하고자 하는 환경을 config file 에 기록하기
  - `application.yaml`, `application.properties`
- Build 후 실행 시점에 config 지정하기
  - `java -jar your-application.jar —spring.config.location=file:/path/to/your/application.yml`

1. **Using Application Properties**:

Set the active profile in the `application.properties` or `application.yml`

properties

```
# application.properties
spring.profiles.active=dev
```

Or in `application.yml`:

yaml

```
# application.yml
spring:
  profiles:
    active: dev
```

Spring 뜯어보기

Inversion of Control

- Spring Bean
  - IoC 로 관리되는 주입 가능한 Object
  - @Configuration class 에 관리될 Bean 을 등록
  - 등록된 Bean 은 요청되는 곳에 Injection

kotlin

```kotlin
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration

interface PaymentService {
    fun processPayment()
}

class CreditCardPaymentService : PaymentService {
    override fun processPayment() {
        println("Processing payment with credit card.")
    }
}

@Configuration
class AppConfig {
    @Bean
    fun paymentService(): PaymentService = CreditCardPaymentService()
}
```

kotlin

```kotlin
@Service
class OrderService @Autowired constructor(private val paymentService: PaymentService) {
    fun placeOrder() {
        paymentService.processPayment()
        println("Order placed successfully.")
    }
}
```

Spring 뜯어보기

Inversion of Control

- 지금까지 사용한 @Component, @Service, @Repository 어노테이션?
- @Bean 으로 등록할 거라고 이야기 해 주는 Stereotype annotation
- + 직접 @Configuration 에 등록하지 않아도, 스프링이 알아서(Component-Scan) 해서 등록
  - Java Reflection 을 사용해서 JVM 의 클래스를 훑어보며…
- @SpringBootApplication 은 내부적으로 ComponentScan 을 사용 중

Spring 뜯어보기

Aspect Oriented Programming

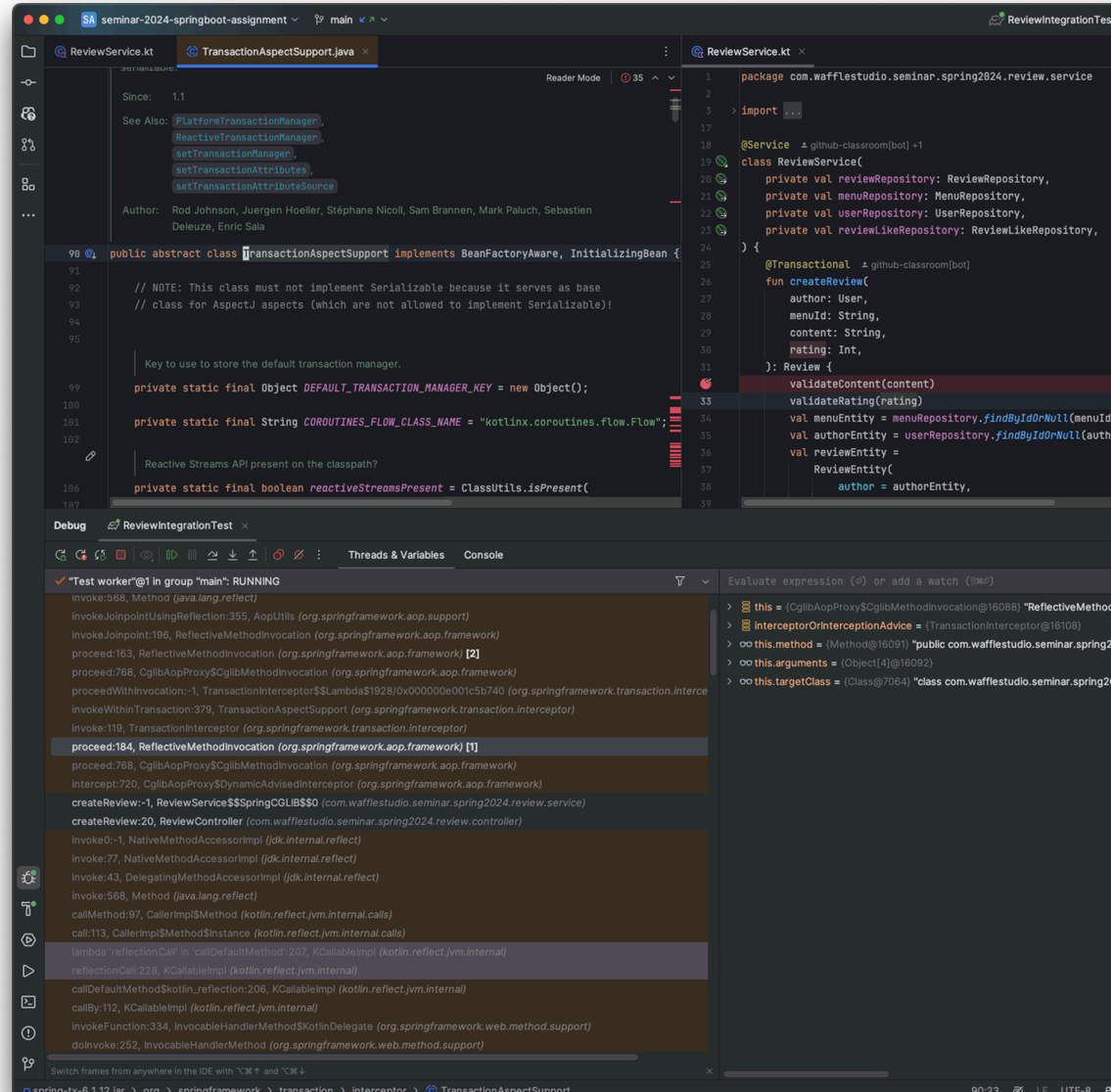
- @Aspect: Cross-cutting 영역을 모듈화 하여 처리
- 특정 메서드의 호출 전후 시점 / Exception 의 발생 시점 등에 특정한 동작을 수행하고 싶다
  - @PointCut -> 어느 표현(어떤 Method 와 같은) 에서 필요한지 기록
  - @Advice -> 어느 시점 (Before, After 등) 에 실행할지 기록
- e.g.) Logging, Transaction 등

Spring 뜯어보기

Aspect Oriented Programming

- 특정 Method 가 Transaction 내부에서 동작하였으면 한다 -> @Transaction

- 내부를 까 보면? CglibProxy 를 사용해 자바 리플렉션으로 메서드를 프록시 메서드로 바꿔버린다



Gradle
빌드 툴이 하는 일

- Java, Kotlin, Android 등 JVM 위에서 굴러가는 프로젝트에 많이 쓰이는 빌드 툴
- 빌드 툴이 하는 일
  - 의존성 관리 (사용하는 라이브러리들의 관리)
  - 컴파일(빌드) 해주기 / 실행해주기
  - 빌드 태스크 관리
- 좋은 빌드 툴
  - 효율적인 빌드 (Incremental Build)
  - 효율적인 태스크 관리

Gradle

Gradle 뜯어보기

- Gradle Script 로 내 프로젝트 빌드 방법을 명시
  - 어떤 의존성을 넣을지
    - e.g. spring-boot
  - 어떤 플러그인을 넣을지
    - e.g. ktlint
  - 어떤 태스크를 추가할지
    - e.g. ktlintFormat

```
tasks.withType<Test> {
    useJUnitPlatform()
}
```

```
plugins {
    id("org.springframework.boot") version "3.3.3"
    id("io.spring.dependency-management") version "1.1.6"
    kotlin("jvm") version "2.0.20"
    kotlin("plugin.spring") version "2.0.20"
    kotlin("plugin.jpa") version "2.0.20"
    id("org.jlleitschuh.gradle.ktlint") version "12.1.1"
}
```

```
dependencies { Edit Starters...
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("org.jetbrains.kotlin:kotlin-reflect")
    implementation("org.mindrot:jbcrypt:0.4")
    implementation("com.mysql:mysql-connector-j:8.2.0")
    implementation("io.jsonwebtoken:jjwt-api:0.11.5")
    implementation("io.jsonwebtoken:jjwt-impl:0.11.5")
    implementation("io.jsonwebtoken:jjwt-jackson:0.11.5")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
}
```

Gradle

Gradle 뜯어보기

- `./gradlew tasks`
  - 실행 가능한 task 들이 나온다
  - Plugin 들이 추가했을 수도,
    우리가 Script 로 추가했을 수도..
  - e.g. classes, jar,
    ktlintFormat, bootRun