라이브 세션-2022.09.06(월)-트랜잭션

✅ 아바타 실습

- 공지 사항 기능 계속

✅ Spring Data JPA에서 리뷰하지 못했던 내용

1️⃣ Fetch 전략

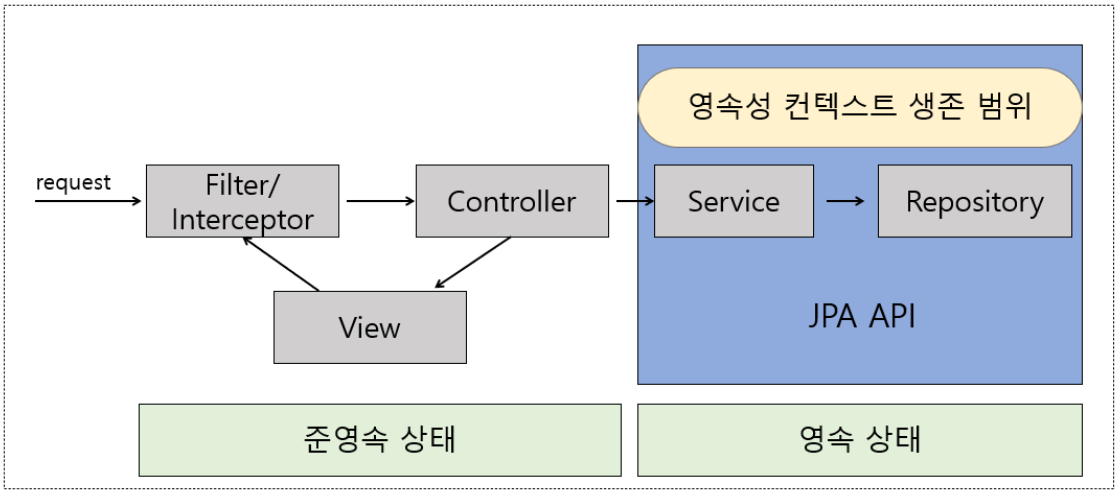
- **Eager**
  - 데이터를 즉시 가져온다.
- **Lazy**
  - 데이터를 필요한 시점에 가져온다.

2️⃣ `jpa.open-in-view: true` 에 대한 설명

- Lazy 로딩을 JPA API 사용 바깥까지 허용하는 것.(default: true)
- false로 하면 서비스 바깥 쪽 영역까지 Lazy 로딩을 허용하지 않는다.

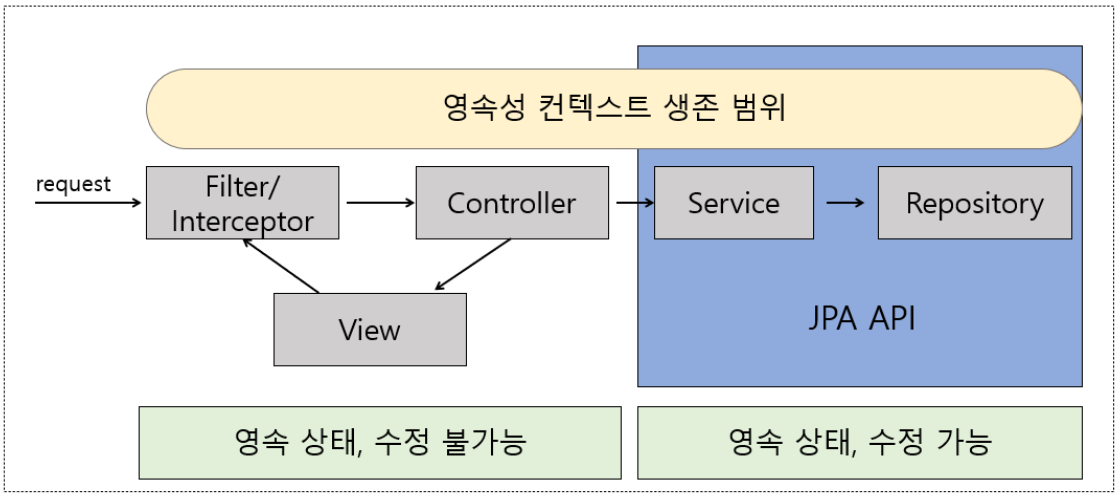
- `jpa.open-in-view: false`

jpa.open-in-view: false



- `jpa.open-in-view: true`

jpa.open-in-view: true

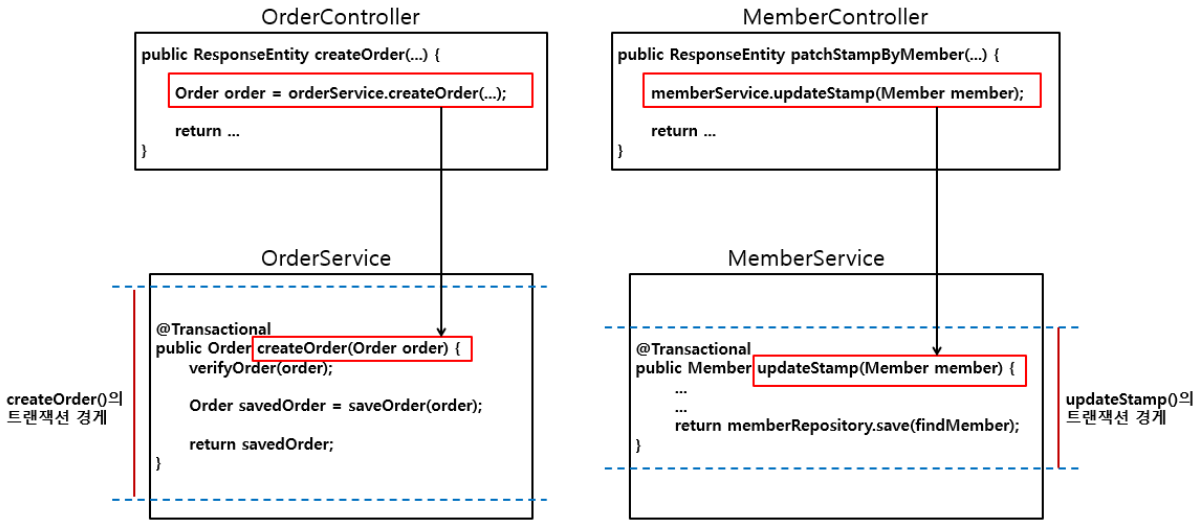


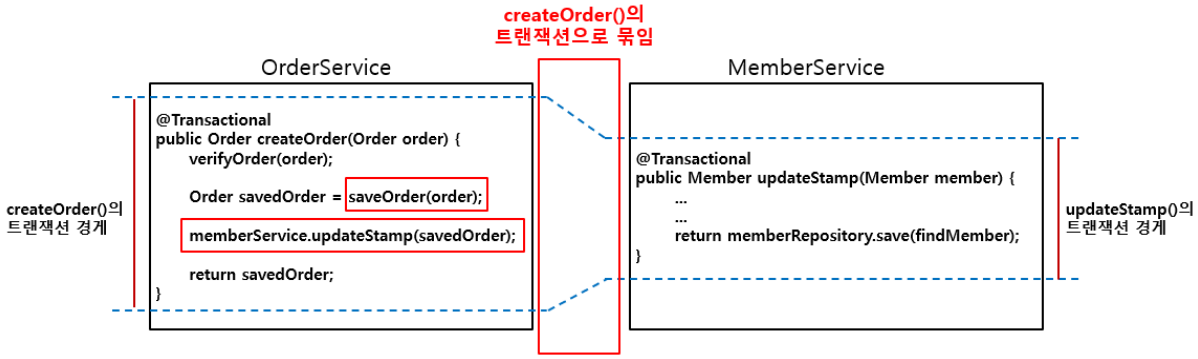
✅ 트랜잭션이란?

- 여러 개의 작업을 하나의 작업으로 묶어서 All or Nothing의 규칙이 적용되어 하는 프로세스

- ACID 원칙
  - 원자성(**A**tomicity)
    - 작업을 더이상 쪼갤 수 없음
  - 일관성(**C**onsistency)
    - 비즈니스 로직에서 의도하는대로 일관성있게 저장되거나 변경되는 것
  - 격리성(**I**solation)
    - 여러 개의 트랜잭션이 실행될 경우 각각 독립적으로 실행이 되어야 함
      - **ThreadLocal**
  - 지속성(**D**urability)
    - 트랜잭션이 완료되면 그 결과는 지속되어야 한다
    - 물리적인 저장소

✅ 트랜잭션의 경계





✅ Spring에서 트랜잭션 적용 방법

- @Transactional 사용
- AOP 방식

✅ 트랜잭션 전파

트랜잭션의 경계에서 진행 중인 **트랜잭션이 존재할 때 또는 존재하지 않을 때, 어떻게 동작할 것인지 결정하는 방식**

- `Propagation.REQUIRED`
  - 진행 중인 트랜잭션이 없으면 새로 시작하고, 진행 중인 트랜잭션이 있으면 해당 트랜잭션에 참여
- `Propagation.REQUIRES_NEW`
  - 진행 중인 트랜잭션이 없으면 새로운 트랜잭션이 시작
- `Propagation.MANDATORY`
  - 진행 중인 트랜잭션이 없으면 예외를 발생
- `Propagation.NEVER`
  - 트랜잭션을 필요로 하지 않음

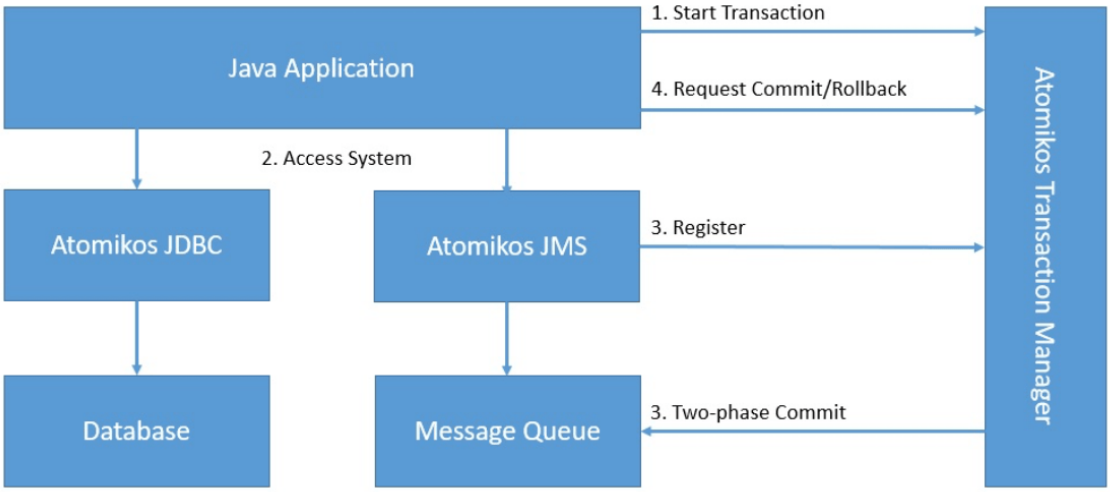
✅ 트랜잭션 격리 레벨

트랜잭션 격리성의 수준을 어떻게 설정할 것인가

- `Isolation.DEFAULT`
  - 데이터베이스에서 제공하는 기본 값
- `Isolation.READ_UNCOMMITTED`
  - 다른 트랜잭션에서 커밋하지 않은 데이터를 읽는 것을 허용
- `Isolation.READ_COMMITTED`
  - 다른 트랜잭션에 의해 커밋된 데이터를 읽는 것을 허용
- `Isolation.REPEATABLE_READ`
  - 트랜잭션 내에서 한 번 조회한 데이터를 반복해서 조회해도 같은 데이터가 조회
- `Isolation.SERIALIZABLE`
  - 동일한 데이터에 대해서 동시에 두 개 이상의 트랜잭션이 수행되지 못하도록 함.

✅ 분산 트랜잭션(지금은 몰라도 됩니다. 가볍게 넘어가세요)

- XA(eXtened Architecture) 프로토콜을 지원하는 Datasource를 사용
  - DB 벤더에서 지원
- 분산 트랜잭션의 종류
  - WAS가 제공하는 JTA(Java Transaction API) 서비스 이용
    - JTA 서버 필요
  - 애플리케이션에 내장된 독립형 JTA 트랜잭션 매니저
    - Atomikos



✅ 실습 과제 Solution 설명

- 코드로 설명

✅ DTO 리팩토링 설명

- DTO 클래스를 static 멤버 클래스로 모으는 작업
- 코드로 설명

✅ 디자인 패턴 소개

- 실습 과제에 포함된 EmailSendable에 대한 **전략 패턴** 설명

디자인 패턴(Design Pattern)이란?

: 과거의 소프트웨어 개발 과정에서 축적된 설계 노하우에 패턴별로 이름을 붙여 재사용하기 좋은 형태로 특정 규칙들을 묶어서 정리한 것

- 객체 지향 설계의 베스트 프랙티스
- 즉, 이미 누군가에 의해 잘 만들어진 객체 지향의 설계 기법
- 객체 간의 관계 : 상속, 인터페이스, 구성(Composition) 등을 사용
- 개발자 및 설계자로 한 단계 성장하기 위한 필수 학습

1️⃣ 빌더 패턴

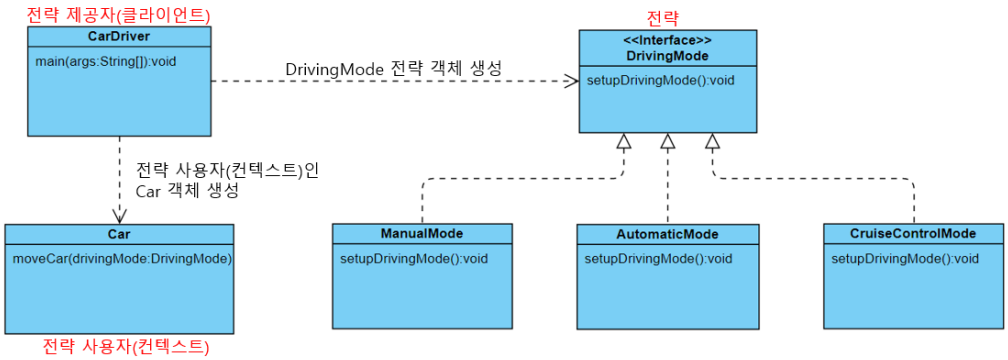
- 생성자로 객체를 생성하지 않고, 빌더를 사용하는 방법
- 특징
  - 필요한 필드만 설정 가능
  - 필드가 추가 되어도 별도의 생성자를 추가로 만들 필요 없음(유연성)
  - 가독성이 좋아짐
  - 불변 객체를 만들어야 할 때 사용
- 코드 확인

2️⃣ 전략 패턴

전략 패턴의 구성 요소

- 전략을 실행하는 **전략 객체** - **변하는 것**
- 전략 객체를 사용하는 **사용자(컨텍스트)** - **변하지 않는것**
- 전략 객체를 사용자(컨텍스트)에게 공급(주입)하는 **클라이언트**

전략 패턴 적용 예 - 클래스 다이어그램



- **예제 코드 확인**

✅ 기타

- CustomBeanUtils 설명