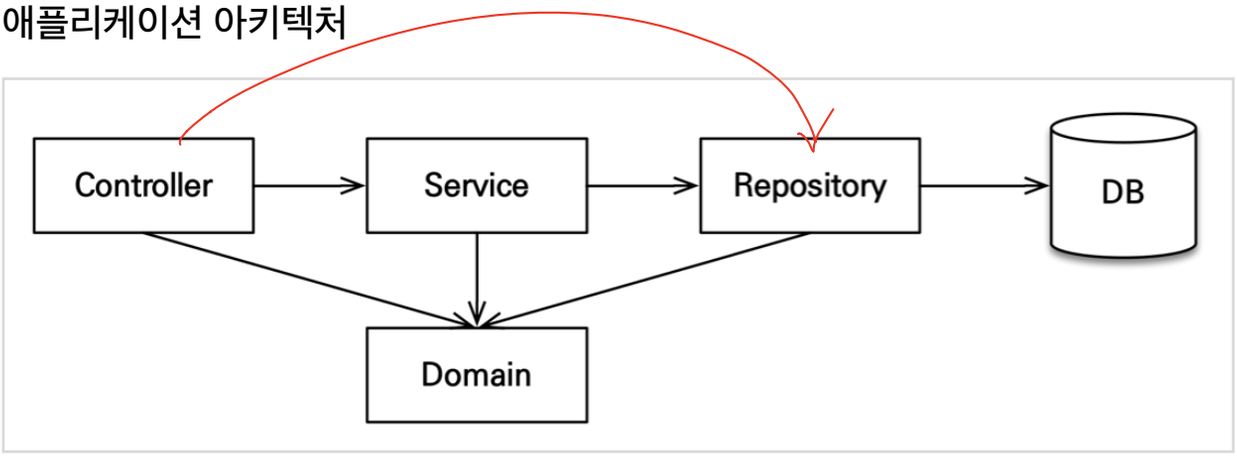
애플리케이션 아키텍처



계층형 구조 사용

- controller, web: 웹 계층
- service: 비즈니스 로직, 트랜잭션 처리
- repository: JPA를 직접 사용하는 계층, 엔티티 매니저 사용
- domain: 엔티티가 모여 있는 계층, 모든 계층에서 사용

패키지 구조

- jpabook.jpashop
  - domain
  - exception
  - repository
  - service
  - web

개발 순서: 서비스, 리포지토리 계층을 개발하고, 테스트 케이스를 작성해서 검증, 마지막에 웹 계층 적용
마지막에 컨트롤러

회원 도메인 개발

구현 기능

- 회원 등록
- 회원 목록 조회

순서

- 회원 엔티티 코드 다시 보기
- 회원 리포지토리 개발
- 회원 서비스 개발
- 회원 기능 테스트

회원 리포지토리 개발

회원 리포지토리 코드

```
package jpabook.jpashop.repository;

import jpabook.jpashop.domain.Member;
import org.springframework.stereotype.Repository;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;

@Repository
public class MemberRepository {

    @PersistenceContext  jpa가 제공하는 표준 애노테이션->em에 알아서 주입해줌
    private EntityManager em;

    public void save(Member member) {
        em.persist(member);
    }

    public Member findOne(Long id) {
        return em.find(Member.class, id);
    }

    public List<Member> findAll() {
        return em.createQuery("select m from Member m", Member.class)
                .getResultList();  jpa쿼리를 작성해야함(jpql)
    }

    public List<Member> findByName(String name) {
        return em.createQuery("select m from Member m where m.name = :name",
Member.class)
                .setParameter("name", name)
                .getResultList();
    }
}
```

@SpringbootApplication가있는 패키지와 하위 패키지는 다 스프링이 컴포넌트스캔을 함.

기술 설명

- `@Repository` : 스프링 빈으로 등록, JPA 예외를 스프링 기반 예외로 예외 변환
- `@PersistenceContext` : 엔티티 메니저(`EntityManager`) 주입
- `@PersistenceUnit` : 엔티티 메니터 팩토리(`EntityManagerFactory`) 주입

기능 설명

- `save()`
- `findOne()`
- `findAll()`
- `findByName()`

회원 서비스 개발

회원 서비스 코드

```
package jpabook.jpashop.service;

import jpabook.jpashop.domain.Member;
import jpabook.jpashop.repository.MemberRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

@Service
@Transactional(readOnly = true)
public class MemberService {

    @Autowired
    MemberRepository memberRepository;

    /**
     * 회원가입
     */
    @Transactional //변경
    public Long join(Member member) {

        validateDuplicateMember(member); //중복 회원 검증
```

JPA에서 데이터변경은 transactional안에서 이루어져야 한다!
readOnly=true로 해놓으면 값을 변경하지 않는 메소드에서는 최적화가 들어간다.(값을 변경하는데 해놓으면 변경이 안된다)

생성자주입을 추천합니다.요즘스프링은 생성자 하나면 알아서 해줌
lombok의 requireArgsConstructor을 쓰면 final로 생성된 필드만 생성자를 만들어줌

pk값인 id 값을 보장해줌 (Spring이)

```
        memberRepository.save(member);
        return member.getId();
    }

    private void validateDuplicateMember(Member member) {
        List<Member> findMembers =          syncronize해줘야함 원래
memberRepository.findByName(member.getName());
        if (!findMembers.isEmpty()) {       count를 이용하는게 더 최적화임.
            throw new IllegalStateException("이미 존재하는 회원입니다.");
        }
    }

    /**
     * 전체 회원 조회
     */
    public List<Member> findMembers() {
        return memberRepository.findAll();
    }

    public Member findOne(Long memberId) {
        return memberRepository.findOne(memberId);
    }
}
```

기술 설명

- `@Service`
- `@Transactional` : 트랜잭션, 영속성 컨텍스트
  - `readOnly=true` : 데이터의 변경이 없는 읽기 전용 메서드에 사용, 영속성 컨텍스트를 플러시 하지 않으므로 약간의 성능 향상(읽기 전용에는 다 적용)
  - 데이터베이스 드라이버가 지원하면 DB에서 성능 향상
- `@Autowired`
  - 생성자 Injection 많이 사용, 생성자가 하나면 생략 가능

기능 설명

- `join()`
- `findMembers()`
- `findOne()`

> **참고:** 실무에서는 검증 로직이 있어도 멀티 쓰레드 상황을 고려해서 회원 테이블의 회원명 컬럼에 유니크 제약 조건을 추가하는 것이 안전하다.

> 참고: 스프링 필드 주입 대신에 생성자 주입을 사용하자.

필드 주입

```
public class MemberService {

    @Autowired
    MemberRepository memberRepository;
    ...
}
```

생성자 주입

```
public class MemberService {

    private final MemberRepository memberRepository;

    public MemberService(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }
    ...
}
```

- 생성자 주입 방식을 권장
- 변경 불가능한 안전한 객체 생성 가능
- 생성자가 하나면, `@Autowired`를 생략할 수 있다.
- `final` 키워드를 추가하면 컴파일 시점에 `memberRepository`를 설정하지 않는 오류를 체크할 수 있다. (보통 기본 생성자를 추가할 때 발견)

lombok

```
@RequiredArgsConstructor
public class MemberService {
```

```
    private final MemberRepository memberRepository;
    ...
}
```

> 참고: 스프링 데이터 JPA를 사용하면 `EntityManager`도 주입 가능

```
@Repository
@RequiredArgsConstructor
public class MemberRepository {

    private final EntityManager em;
    ...
}
```

`MemberService` 최종 코드

```
@Service
@Transactional(readOnly = true)
@RequiredArgsConstructor
public class MemberService {

    private final MemberRepository memberRepository;

    /**
     * 회원가입
     */
    @Transactional //변경
    public Long join(Member member) {

        validateDuplicateMember(member); //중복 회원 검증
        memberRepository.save(member);
        return member.getId();
    }

    private void validateDuplicateMember(Member member) {
        List<Member> findMembers =
```

```
memberRepository.findByName(member.getName());
        if (!findMembers.isEmpty()) {
            throw new IllegalStateException("이미 존재하는 회원입니다.");
        }
    }

    /**
     * 전체 회원 조회
     */
    public List<Member> findMembers() {
        return memberRepository.findAll();
    }

    public Member findOne(Long memberId) {
        return memberRepository.findOne(memberId);
    }
}
```

회원 기능 테스트

테스트 요구사항

- 회원가입을 성공해야 한다.
- 회원가입 할 때 같은 이름이 있으면 예외가 발생해야 한다.

회원가입 테스트 코드

```
package jpabook.jpashop.service;

import jpabook.jpashop.domain.Member;
import jpabook.jpashop.repository.MemberRepository;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;
```

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

@RunWith(SpringRunner.class)
@SpringBootTest
@Transactional
public class MemberServiceTest {

    @Autowired MemberService memberService;
    @Autowired MemberRepository memberRepository;

    @Test
    public void 회원가입() throws Exception {

        //Given
        Member member = new Member();
        member.setName("kim");

        //When
        Long saveId = memberService.join(member);

        //Then
        assertEquals(member, memberRepository.findOne(saveId));
    }
```

이거있으면 try catch문 안써도돼

```java
    @Test(expected = IllegalStateException.class)
    public void 중복_회원_예외() throws Exception {
        //Given
        Member member1 = new Member();
        member1.setName("kim");

        Member member2 = new Member();
        member2.setName("kim");

        //When
        memberService.join(member1);
        memberService.join(member2); //예외가 발생해야 한다.

        //Then
```

```
            fail("예외가 발생해야 한다.");
        }
    }
```

기술 설명

- `@RunWith(SpringRunner.class)` : 스프링과 테스트 통합 스프링랑 같이 엮어서 실행하기위해
- `@SpringBootTest` : 스프링 부트 띄우고 테스트(이게 없으면 `@Autowired` 다 실패)
- `@Transactional` : 반복 가능한 테스트 지원, 각각의 테스트를 실행할 때마다 트랜잭션을 시작하고 **테스트가 끝나면 트랜잭션을 강제로 롤백** (이 어노테이션이 테스트 케이스에서 사용될 때만 롤백)

기능 설명

- 회원가입 테스트
- 중복 회원 예외처리 테스트

> 참고: 테스트 케이스 작성 고수 되는 마법: Given, When, Then
> (http://martinfowler.com/bliki/GivenWhenThen.html)
> 이 방법이 필수는 아니지만 이 방법을 기본으로 해서 다양하게 응용하는 것을 권장한다.

테스트 케이스를 위한 설정

테스트는 케이스 격리된 환경에서 실행하고, 끝나면 데이터를 초기화하는 것이 좋다. 그런 면에서 메모리 DB를 사용하는 것이 가장 이상적이다.
추가로 테스트 케이스를 위한 스프링 환경과, 일반적으로 애플리케이션을 실행하는 환경은 보통 다르므로 설정 파일을 다르게 사용하자.
다음과 같이 간단하게 테스트용 설정 파일을 추가하면 된다.

- `test/resources/application.yml`

main과 test의 resources/applicationyml을 달리하면 각자의 것으로 됨.다른쪽것은 무시됨.

```
spring:
#  datasource:
#    url: jdbc:h2:mem:testdb
#    username: sa
#    password:
#    driver-class-name: org.h2.Driver

#  jpa:
#    hibernate:
```

```
#      ddl-auto: create
#    properties:
#      hibernate:
         #        show_sql: true
#         format_sql: true
#    open-in-view: false

logging.level:
  org.hibernate.SQL: debug
#  org.hibernate.type: trace
```

이제 테스트에서 스프링을 실행하면 이 위치에 있는 설정 파일을 읽는다.
(만약 이 위치에 없으면 `src/resources/application.yml` 을 읽는다.)

스프링 부트는 datasource 설정이 없으면, 기본적을 메모리 DB를 사용하고, driver-class도 현재 등록된 라이브러리를 보고 찾아준다. 추가로 `ddl-auto` 도 `create-drop` 모드로 동작한다. 따라서 데이터소스나, JPA 관련된 별도의 추가 설정을 하지 않아도 된다.

상품 도메인 개발

구현 기능

- 상품 등록
- 상품 목록 조회
- 상품 수정

순서

- 상품 엔티티 개발(비즈니스 로직 추가)
- 상품 리포지토리 개발
- 상품 서비스 개발
- 상품 기능 테스트

상품 엔티티 개발(비즈니스 로직 추가)

상품 엔티티 코드

```
package jpabook.jpashop.domain.item;
```