공통 인터페이스 기능

Created By	(영) 영교
Class	실전! 스프링데이터 JPA
Tags	JPA SpringData
# index	3

순수 JPA기반 리포지토리 만들기
    테스트해보기
공통 인터페이스 설정
공통 인터페이스 적용
공통 인터페이스 분석
    공통 인터페이스 구성
    주의사항
    제네릭 타입
    주요 메서드

순수 JPA기반 리포지토리 만들기

→ 우선 순수한 JPA기반 리포지토리를 만들어 볼 것입니다.

→ 기본 CRUD 기능을 구현할 것입니다.

1. **Create(저장)**

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

2. **Read(조회)**

```
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();
}

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

```
            .getResultList();
}
```

3. **Update(변경 → 변경감지 사용)**
    - JPA에서 변경은 따로 메서드를 이용해서 하지 않는다. 변경감지를 이용하면 트랜잭션내에서 객체으 속성이 변경되면 트랜잭션 종료 전 JPA에서는 변경 감지를 하여 업데이트를 해준다.
4. **Delete(삭제)**

```
public void delete(Long id){
    Member findMember = em.find(Member.class, id);
    em.remove(findMember);
}
```

테스트해보기

▼ Code

```
package study.datajpa.repository;

import org.junit.jupiter.api.Test;
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;
import study.datajpa.entity.Member;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
@Transactional
public class MemberJpaRepositoryTest {
    @Autowired
    MemberJpaRepository memberJpaRepository;

    @Test
    public void testMember() {
        Member member = new Member("memberA");
        Member savedMember = memberJpaRepository.save(member);
        Member findMember = memberJpaRepository.find(savedMember.getId());
        assertThat(findMember.getId()).isEqualTo(member.getId());
        assertThat(findMember.getUsername()).isEqualTo(member.getUsername());
        assertThat(findMember).isEqualTo(member); //JPA 엔티티 동일성 보장
    }

    @Test
    public void basicCRUD() {
        Member member1 = new Member("member1");
        Member member2 = new Member("member2");
```

```
        memberJpaRepository.save(member1);
        memberJpaRepository.save(member2);
        //단건 조회 검증
        Member findMember1 = memberJpaRepository.findById(member1.getId()).get();
        Member findMember2 = memberJpaRepository.findById(member2.getId()).get();
        assertThat(findMember1).isEqualTo(member1);
        assertThat(findMember2).isEqualTo(member2);
        //리스트 조회 검증
        List<Member> all = memberJpaRepository.findAll();
        assertThat(all.size()).isEqualTo(2);

        //카운트 검증
        long count = memberJpaRepository.count();
        assertThat(count).isEqualTo(2);
        //삭제 검증 memberJpaRepository.delete(member1); memberJpaRepository.delete(member2);
        long deletedCount = memberJpaRepository.count();
        assertThat(deletedCount).isEqualTo(0);
    }
}
```

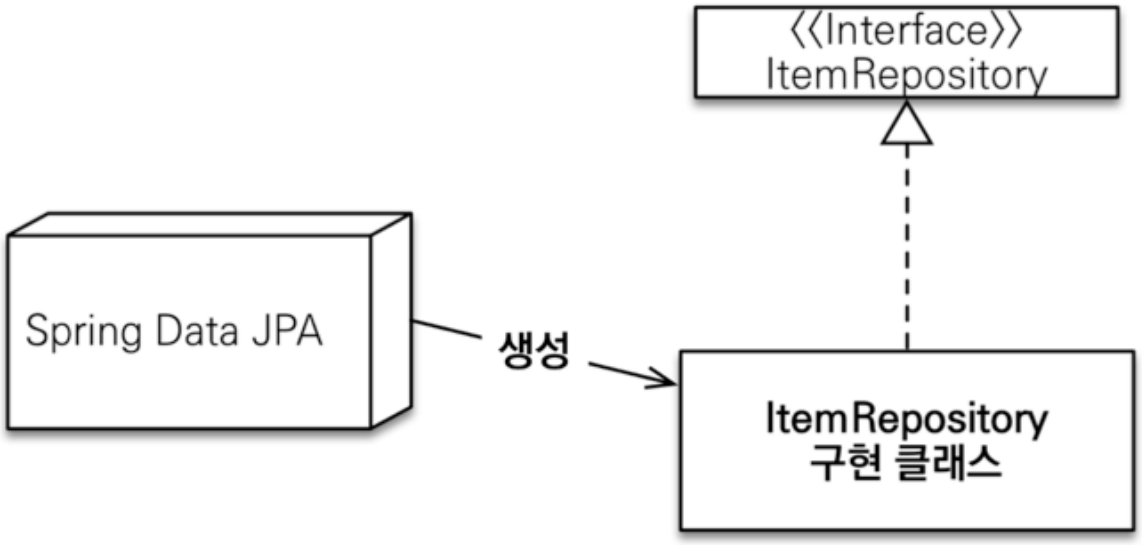
공통 인터페이스 설정

1. JavaConfig 설정
    - 프로젝트명Application.java 파일에 basePackages 설정

```
@Configuration
@EnableJpaRepositories(basePackages = "jpabook.jpashop.repository")
public class AppConfig {}
```

- 스프링 부트 사용 시 `@EnableJpaRepositories(basePackages="jpabook.jpashop.repository")`
  를 따로 작성해줄 필요가 없고 굳이 다른 Config 클래스를 사용할 때 작성해준다.

2. Spring data JPA가 구현 클래스 대신 생성



- 개발자가 Interface만 선언해주면 Spring data가 자동으로 구현클래스를 만들어서 사용한다.
  - Ex: xxxRepository.save() , find(), findAll() ...

xxxRepository.getClass() → class com.sun.proxy.$ProxyXXX

- `@Repository` 어노테이션 생략 가능
  - 컴포넌트 스캔을 스프링 데이터 JPA가 자동으로 처리
  - JPA예외를 스프링 예외로 변환하는 과정도 자동으로 처리

공통 인터페이스 적용

→ 순수 JPA로 구현한 `MemberJpaRepository` 대신 스프링 데이터 JPA가 제공하는 공통 인터페이스 사용

- 스프링 데이터 JPA기반 MemberRepository

```
package study.querydsl.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import study.querydsl.entity.Member;

import java.util.List;

public interface MemberRepository extends JpaRepository<Member, Long> {

    //select m from Member m where m.username = ?;
```

```
    List<Member> findByUsername(String username);
}
```

- `MemberRepository` 테스트 코드

▼ Code

```
package study.querydsl.repository;

import com.querydsl.jpa.impl.JPAQueryFactory;
import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.transaction.annotation.Transactional;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;
import study.querydsl.entity.Member;
import study.querydsl.entity.Team;

import javax.persistence.EntityManager;
import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
@Transactional
class MemberRepositoryTest {
    @Autowired
    EntityManager em;
    @Autowired
    JPAQueryFactory jpaQueryFactory;
    @Autowired
    MemberRepository memberRepository;

    @Test
    public void testMember() {
        Member member = new Member("memberA");
        Member savedMember = memberRepository.save(member);
        Member findMember = memberRepository.findById(savedMember.getId()).get();
        assertThat(findMember.getId()).isEqualTo(member.getId());
        assertThat(findMember.getUsername()).isEqualTo(member.getUsername());

        assertThat(findMember).isEqualTo(member); //JPA 엔티티 동일성 보장
    }

    @Test
    public void basicCRUD() {
        Member member1 = new Member("member1");
        Member member2 = new Member("member2");
        memberRepository.save(member1);
        memberRepository.save(member2);
```

```
        //단건 조회 검증
        Member findMember1 = memberRepository.findById(member1.getId()).get();
        Member findMember2 = memberRepository.findById(member2.getId()).get();
        assertThat(findMember1).isEqualTo(member1);
        assertThat(findMember2).isEqualTo(member2);

        //리스트 조회 검증
        List<Member> all = memberRepository.findAll();
        assertThat(all.size()).isEqualTo(2);

        //카운트 검증
        long count = memberRepository.count();
        assertThat(count).isEqualTo(2);

        //삭제 검증 memberRepository.delete(member1); memberRepository.delete(member2);
        long deletedCount = memberRepository.count();
        assertThat(deletedCount).isEqualTo(0);
    }
}
```

공통 인터페이스 분석

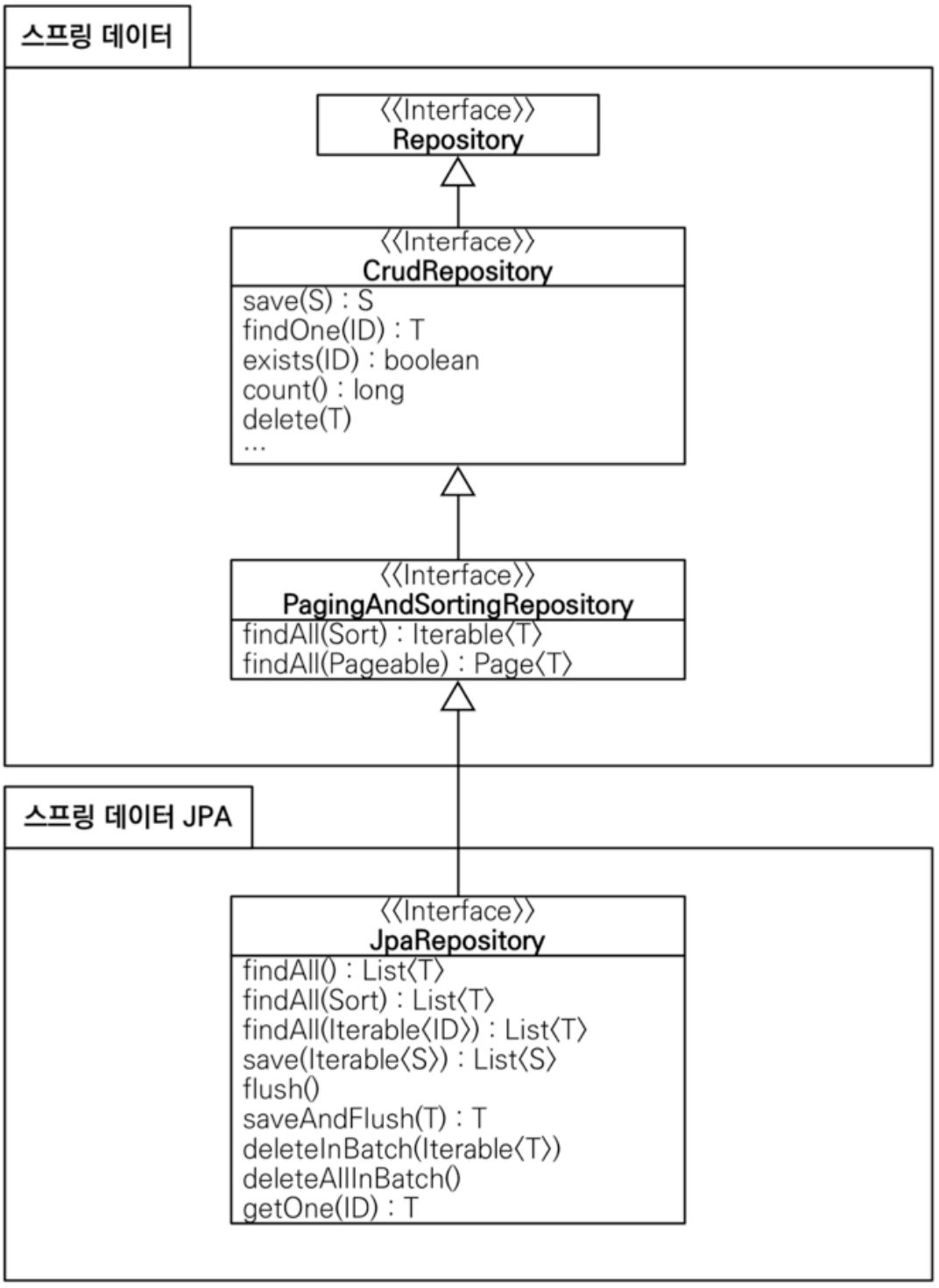
- JpaRepository 인텊이스: 공통 CRUD 제공

  →Ex: find, findAll, save, delete ...

- 제네릭은 <엔티티 타입 , 식별자 타입> 설정

```
public interface ExampleRepository extends JpaRepository<Entity, PrimityType>{}
```

공통 인터페이스 구성



주의사항

- `T findOne(ID)` → `Optional<T> findById(ID)` 로 변경되었다.

제네릭 타입

- `T` :엔티티

- `ID` : 엔티티의 식별자 타입
- `S` : 엔티티와 그 자식 타입

주요 메서드

- `save(S)` : 새로운 엔티티는 저장하고 이미 있는 엔티티는 병합한다.
- `delete(T)` : 엔티티 하나를 삭제한다. 내부에서 EntityManager.remove() 호출
- `findById(ID)` : 엔티티 하나를 조회한다. 내부에서 EntityManager.find() 호출
- `getOne(ID)` : 엔티티를 프록시로 조회한다. 내부에서 EntityManager.getReference() 호출
- `findAll(...)` : 모든 엔티티를 조회한다. 정렬(`Sort`)이나 페이징(`Pageable`) 조건을 파라미터로 제공 가능하다.
