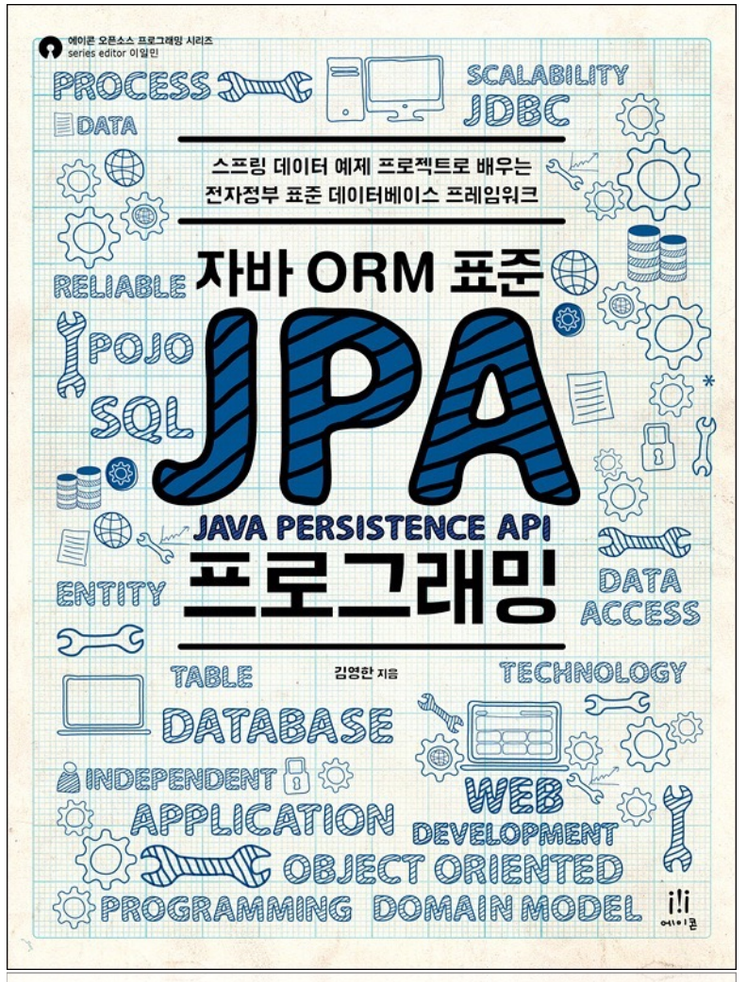
스프링과 JPA

김영한

SI, J2EE 강사, DAUM, SK 플래닛

저서: 자바 ORM 표준 **JPA** 프로그래밍



목차

- 스프링과 JPA
- 스프링 데이터 JPA
- 실무 경험

스프링과 JPA

실습 ex09-spring

스프링과 JPA

- spring-orm 라이브러리 추가
- spring boot는 spring-boot-starter-data-jpa
- LocalContainerEntityManagerFactoryBean 빈 등록
- persistence.xml 필요 없음

스프링과 JPA 환경 설정

```
@Bean
public DataSource dataSource() {
  DriverManagerDataSource dataSource =
                                  new DriverManagerDataSource();
  dataSource.setDriverClassName("org.h2.Driver");
  dataSource.setUrl("jdbc:h2:tcp://localhost/~/test");
  dataSource.setUsername("sa");
  dataSource.setPassword("");
  return dataSource;
}

@Bean
public JpaTransactionManager transactionManager() {
  return new JpaTransactionManager();
}
```

스프링과 JPA 환경 설정

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {

    LocalContainerEntityManagerFactoryBean factoryBean =
                        new LocalContainerEntityManagerFactoryBean();
    factoryBean.setDataSource(dataSource());
    factoryBean.setPackagesToScan("hellojpa.entity");
    factoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

    Properties jpaProperties = new Properties();
    jpaProperties.put(AvailableSettings.SHOW_SQL, true);
    jpaProperties.put(AvailableSettings.FORMAT_SQL, true);
    jpaProperties.put(AvailableSettings.USE_SQL_COMMENTS, true);
    jpaProperties.put(AvailableSettings.HBM2DDL_AUTO, "create");
    jpaProperties.put(AvailableSettings.DIALECT,
                                    "org.hibernate.dialect.H2Dialect");
    jpaProperties.put(AvailableSettings.USE_NEW_ID_GENERATOR_MAPPINGS,
                                                        "true");

    factoryBean.setJpaProperties(jpaProperties);
    return factoryBean;
}
```

엔티티 매니저 주입 - @PersistenceContext

```
@Repository
public class MemberRepository {

  @PersistenceContext
  EntityManager em;
  …
```

실습 ex09-spring

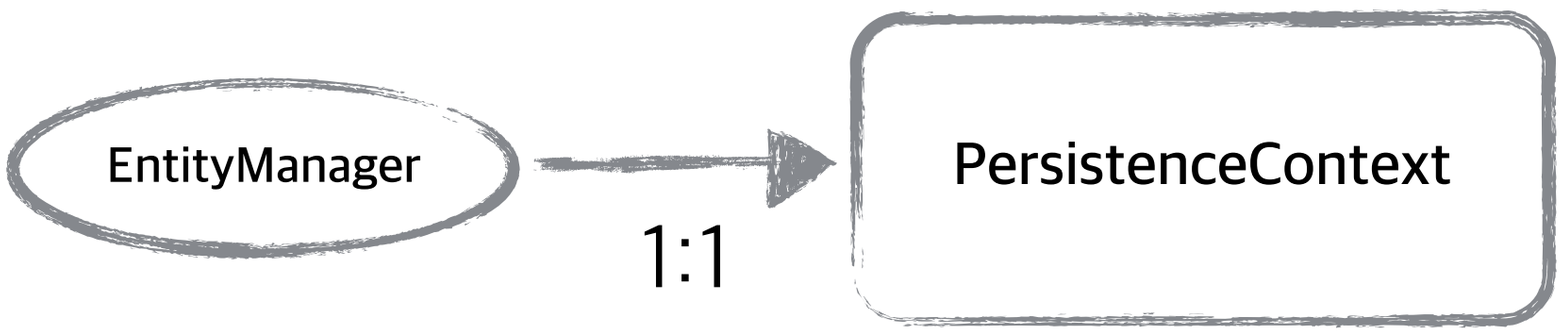
- 스프링으로 설정
- DataSource, JpaTransactionManager
- LocalContainerEntityManagerFactoryBean 추가
- Repository 추가

트랜잭션 범위의 영속성 컨텍스트

- J2EE, 스프링 컨테이너의 기본 전략
- 트랜잭션의 범위와 영속성 컨텍스트의 생존 범위가 같음
- 같은 트랜잭션 안에서는 항상 같은 영속성 컨텍스트에 접근

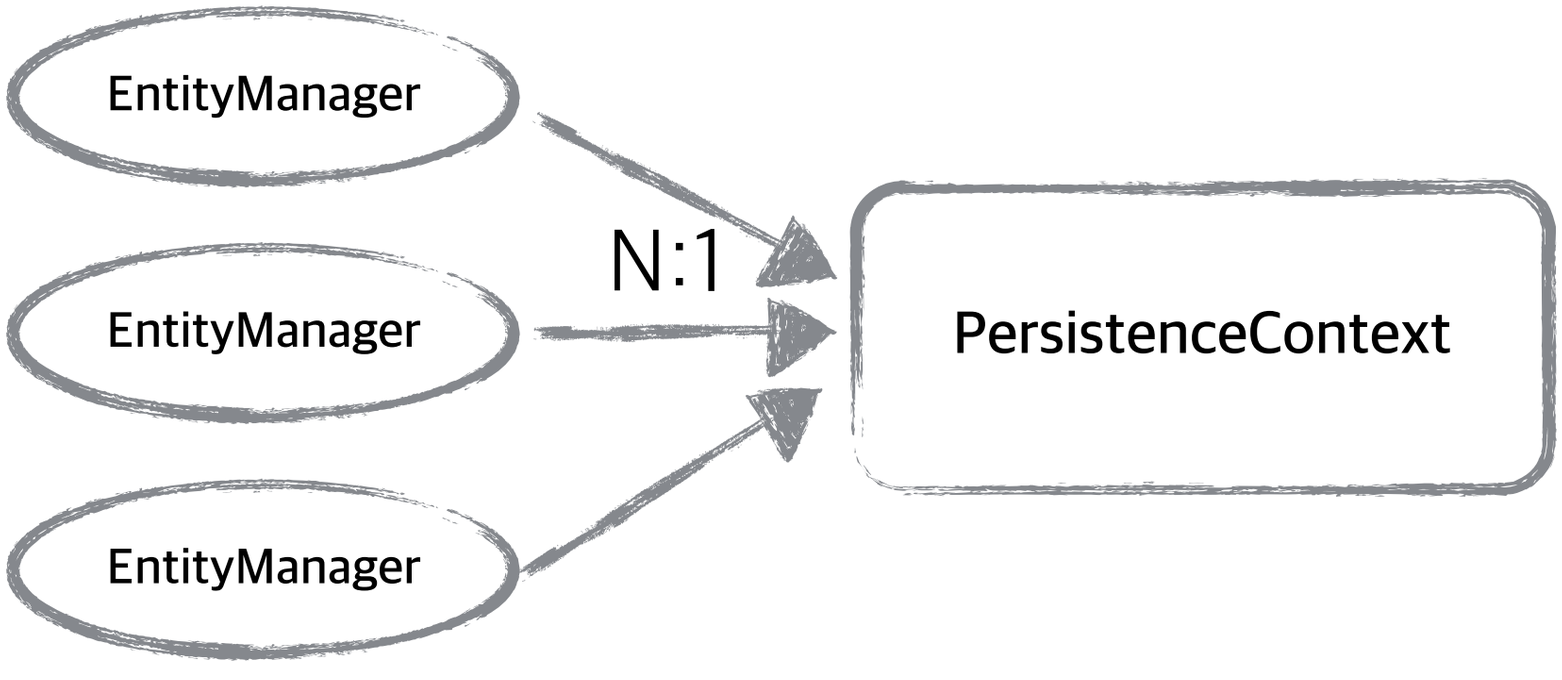
J2SE 환경

엔티티 매니저와 영속성 컨텍스트가 1:1

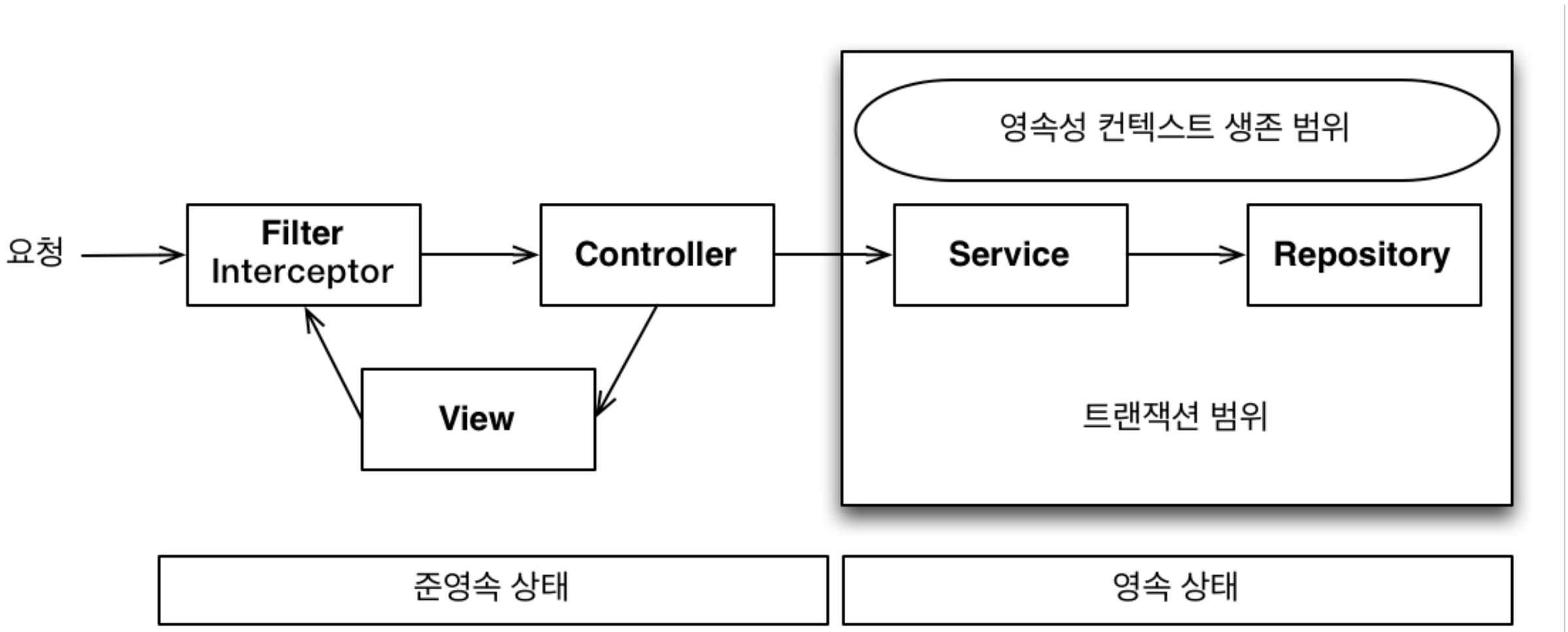


J2EE, 스프링 프레임워크 같은 컨테이너 환경

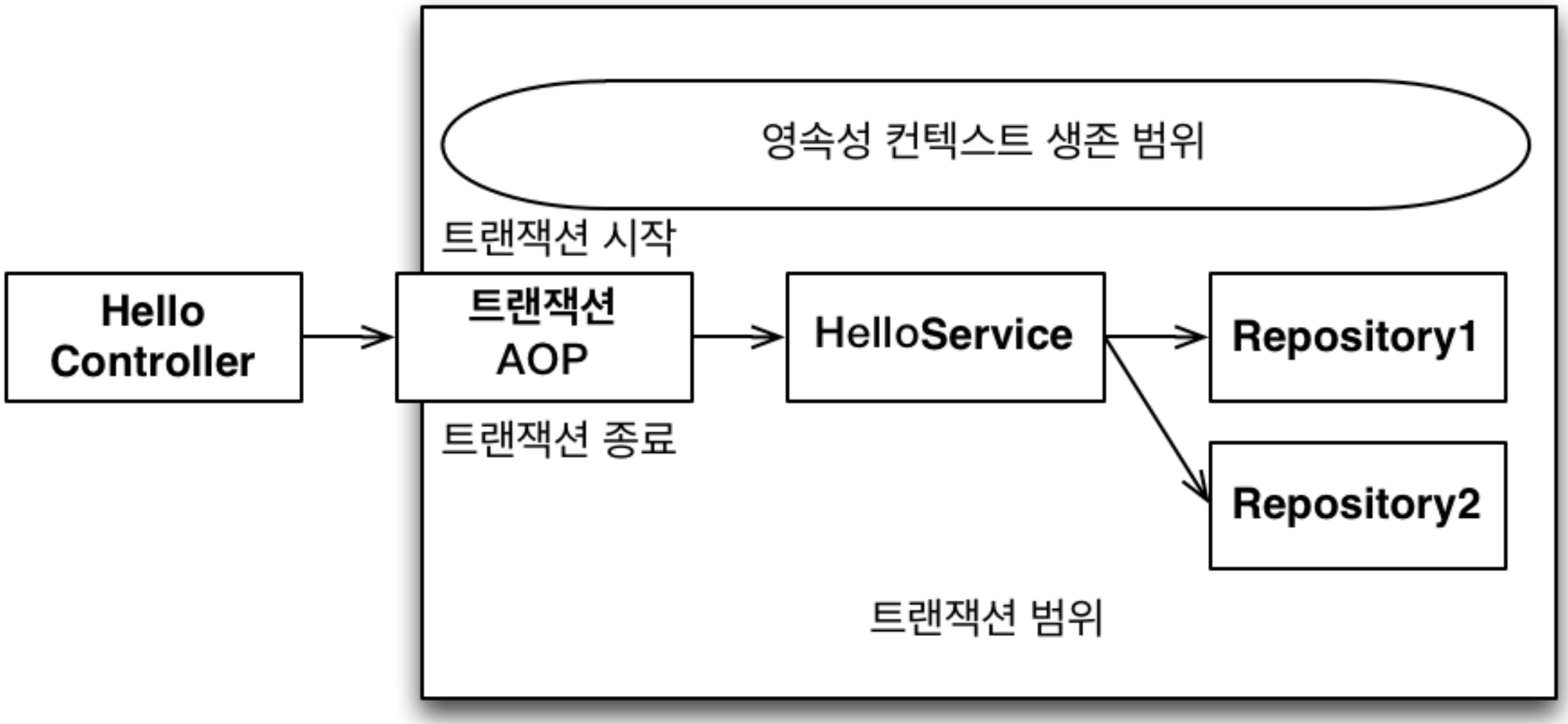
엔티티 매니저와 영속성 컨텍스트가 N:1



트랜잭션 범위의 영속성 컨텍스트



예제


영속성 컨텍스트 생존 범위
트랜잭션 시작
Hello Controller
트랜잭션 AOP
HelloService
Repository1
Repository2
트랜잭션 종료
트랜잭션 범위

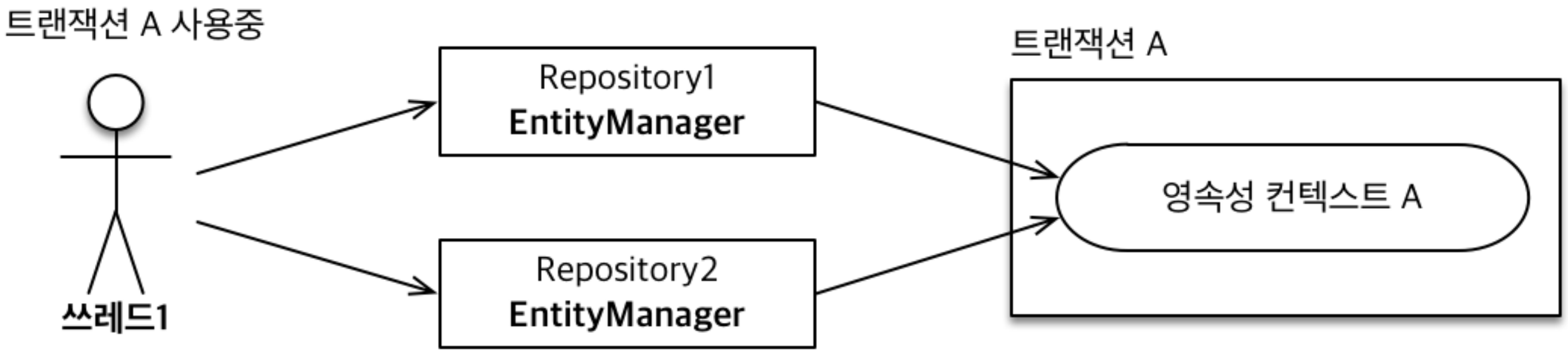
```
@Controller
class HelloController {

    @Autowired HelloService helloService;

    public void hello() {
     //반환된 `member` 엔티티는 준영속 상태다.
     Member member = helloService.logic();
    }
}
```

```java
@Service
class HelloService {

    @Autowired Repository1 repository1;
    @Autowired Repository2 repository2;

    //트랜잭션 시작
    @Transactional
    public void logic() {
     repository1.hello();
     //`member`는 영속 상태다.
     Member member = repository2.findMember();
     return member;
    }
    //트랜잭션 종료
}
```

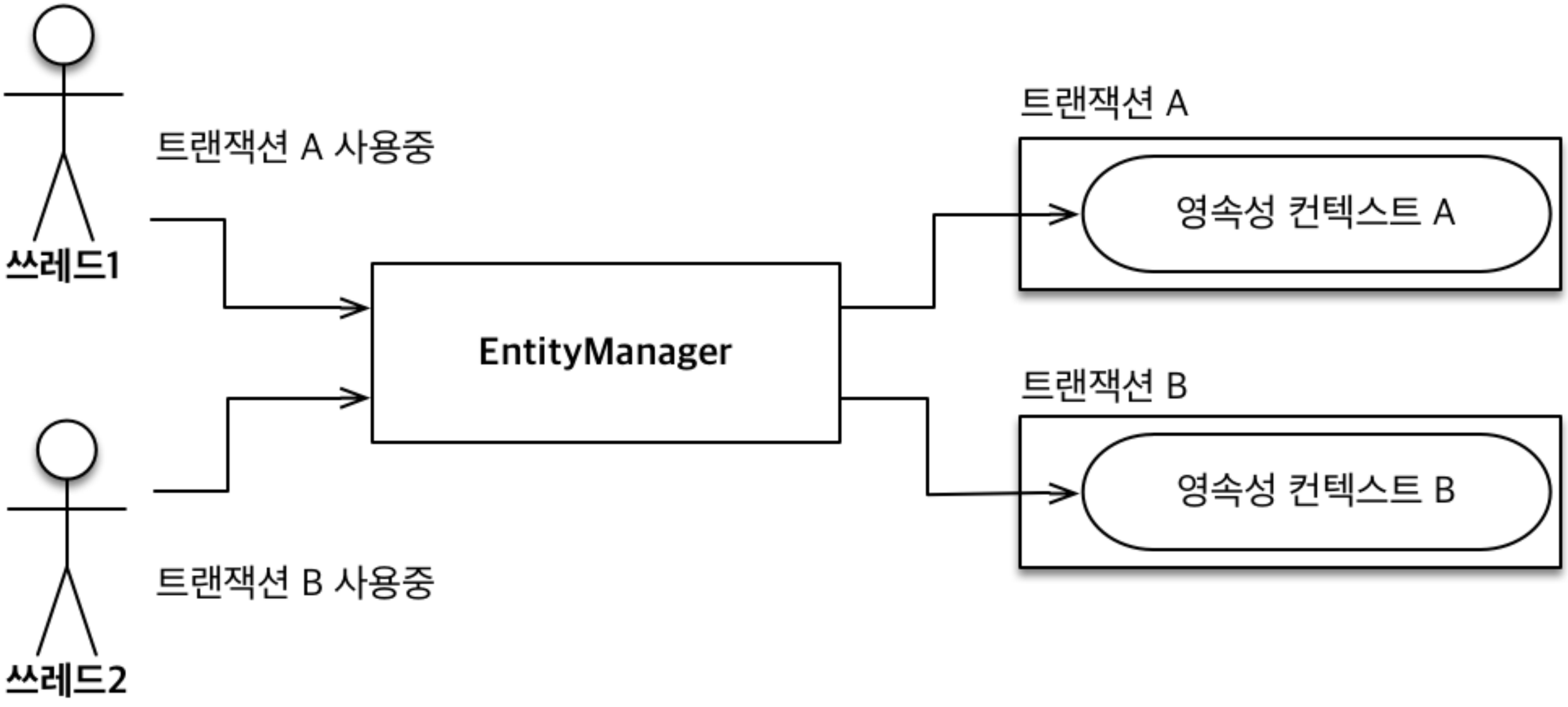
```java
@Repository
class Repository1 {

    @PersistenceContext
    EntityManager em;

    public void hello() {
        em.xxx(); //A. 영속성 컨텍스트 접근
    }
}
```

```java
@Repository
class Repository2 {

    @PersistenceContext
    EntityManager em;

    public Member findMember() {
        return em.find(Member.class, "id1"); //B. 영속성 컨텍스트 접근
    }
}
```

트랜잭션이 같으면
같은 영속성 컨텍스트 사용



트랜잭션이 다르면
다른 영속성 컨텍스트 사용



트랜잭션 범위의 영속성 컨텍스트 정리

- 트랜잭션과 복잡한 멀티 쓰레드 상황을 컨테이너가 처리
- 개발자는 싱글 쓰레드 애플리케이션처럼 단순하게 개발

트랜잭션 범위의 영속성 컨텍스트 약점

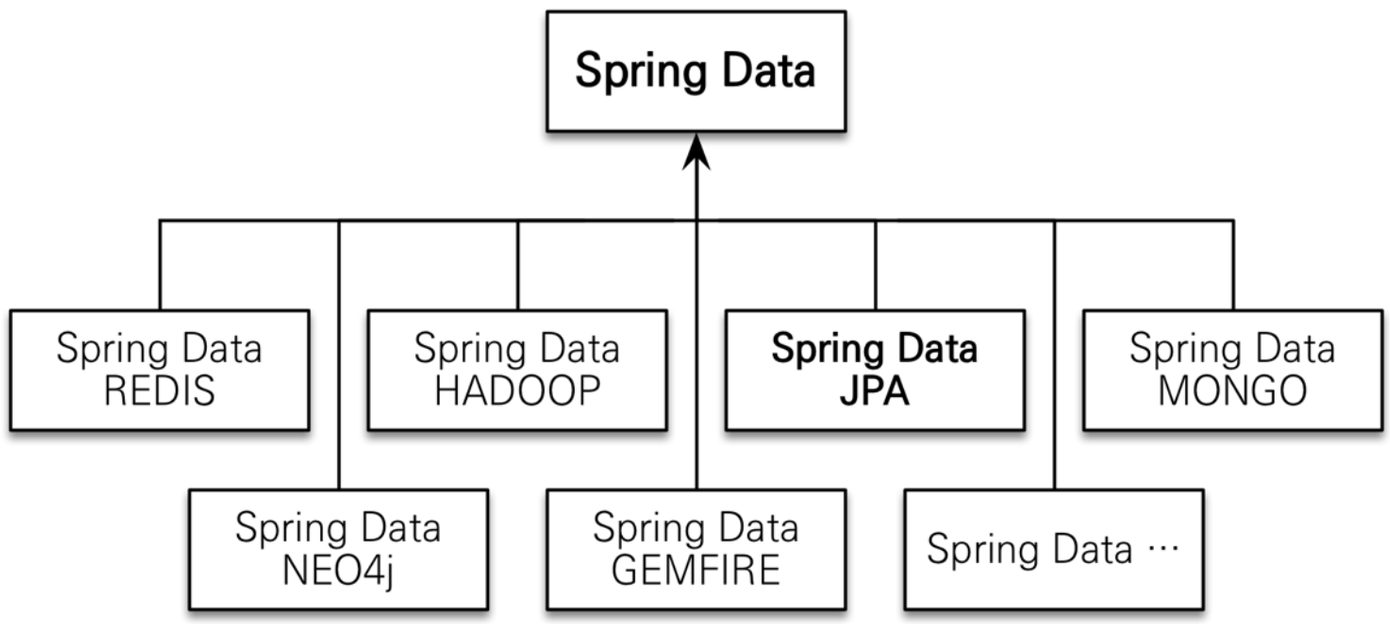
- 트랜잭션이 끝나면 영속성 컨텍스트가 종료
- 엔티티는 준영속 상태가 됨
- 트랜잭션이 끝난 프레젠테이션 계층에서 **지연 로딩 못함**

스프링 데이터 JPA

ex10-springdatajpa

스프링 데이터

- 다양한 데이터 저장소에 대한 접근을 추상화
- 개발자의 편의 제공
- 지루하게 반복하는 데이터 접근 코드를 줄여줌



반복되는 CRUD

```
public class MemberRepository {

    @PersistenceContext
    EntityManager em;

    public void save(Member member) {...}
    public Member findOne(Long id) {...}
    public List<Member> findAll() {...}

    public Member findByUsername(String username) {...}

}

public class ItemRepository {

    @PersistenceContext
    EntityManager em;

    public void save(Item item) {...}
    public Member findOne(Long id) {...}
    public List<Member> findAll() {...}
}
```

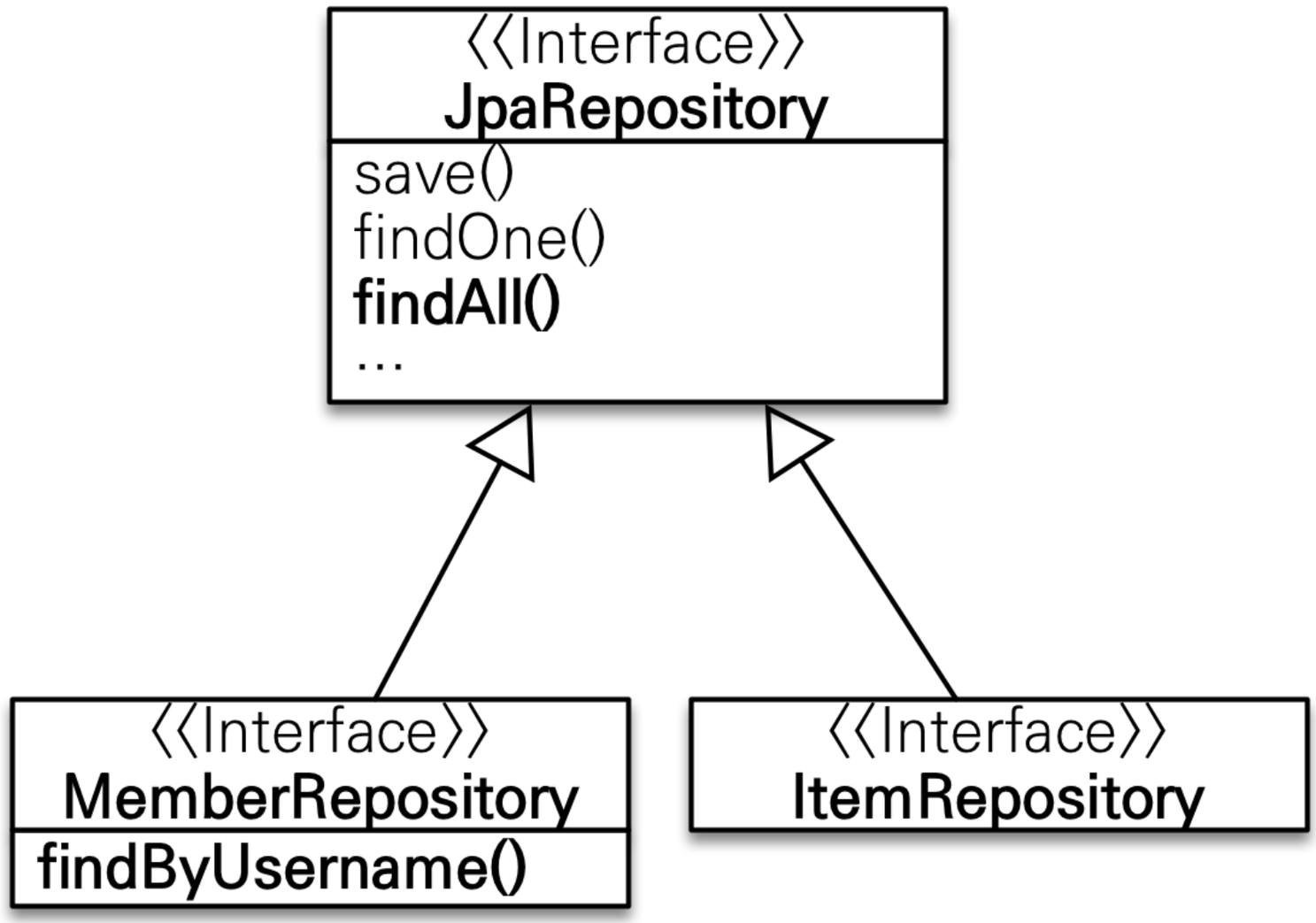
스프링 데이터 JPA 소개

- 지루하게 반복되는 CRUD 문제를 세련된 방법으로 해결
- 개발자는 인터페이스만 작성
- 스프링 데이터 JPA가 구현 객체를 동적으로 생성해서 주입

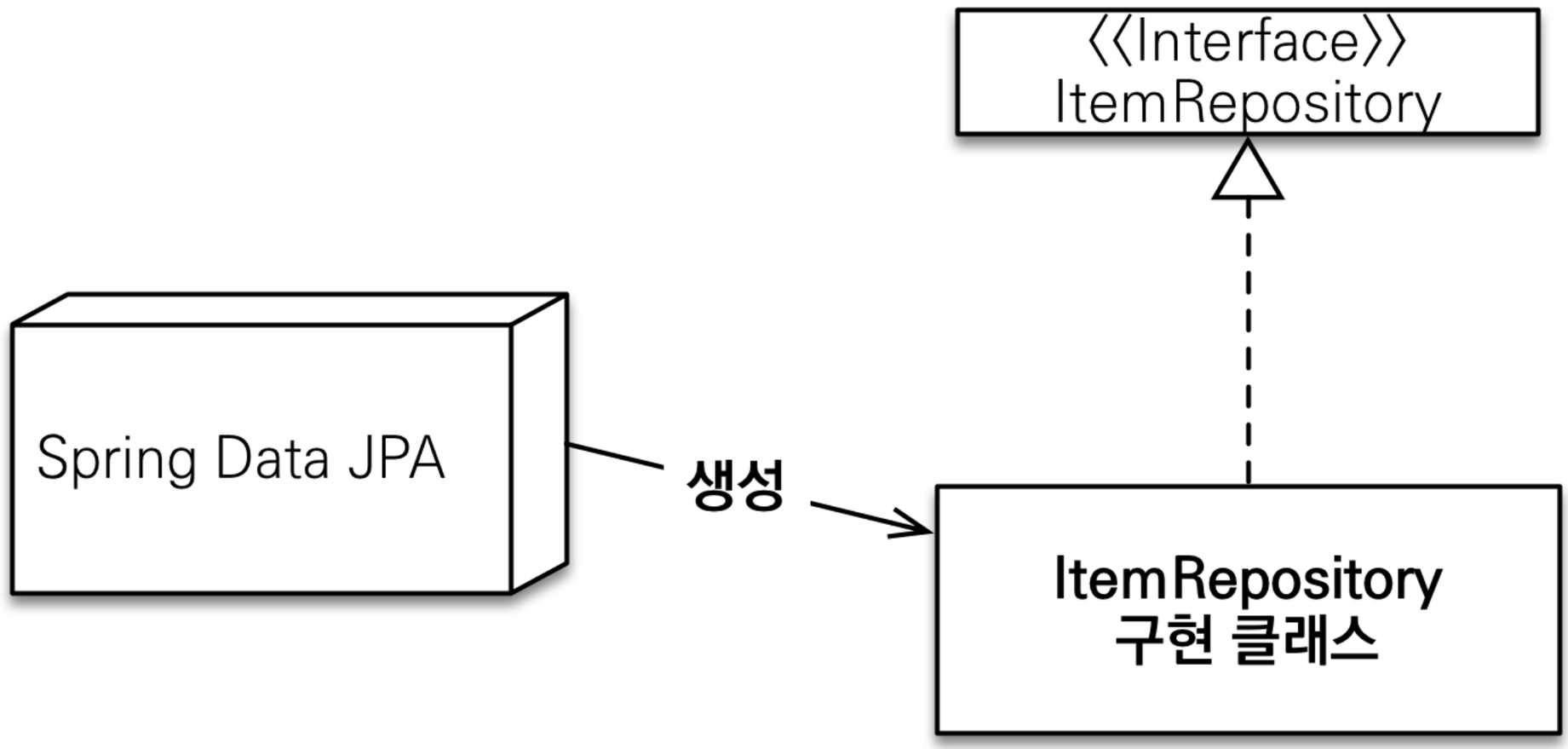
스프링 데이터 JPA 적용

```
public interface MemberRepository extends JpaRepository<Member, Long>{
    Member findByUsername(String username);
}

public interface ItemRepository extends JpaRepository<Item, Long> {
}
```

스프링 데이터 JPA 적용 후 클래스 다이어그램



스프링 데이터 JPA가 구현 클래스 생성



스프링 데이터 JPA 설정 - 라이브러리

```
<!-- 스프링 데이터 JPA -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.8.0.RELEASE</version>
</dependency>
```

```
<!-- 스프링 부트는 다음과 같이 설정 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

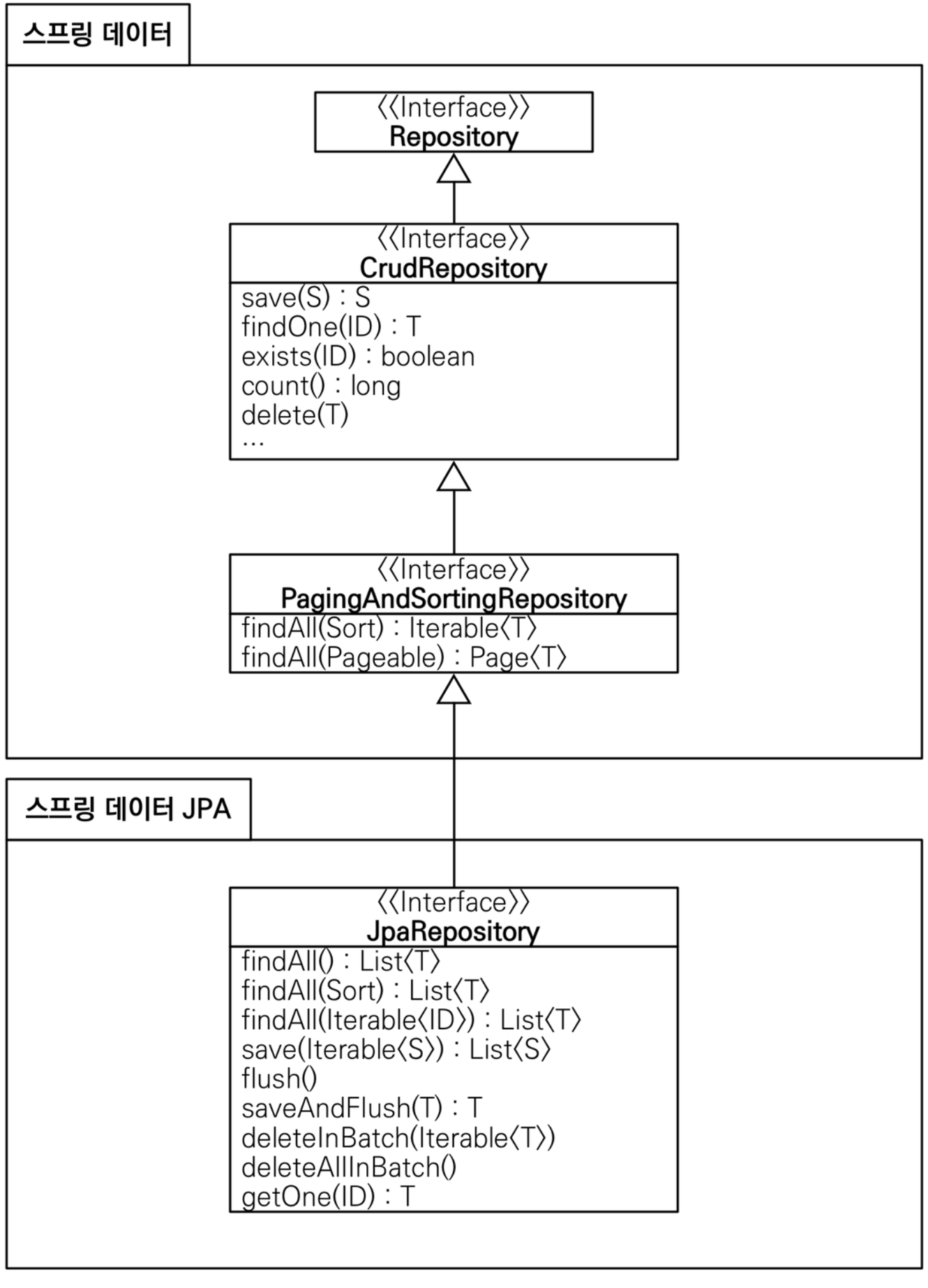
스프링 데이터 JPA 설정 - 환경설정

```
@EnableJpaRepositories(basePackages = “hellojpa.repository")
```

공통 인터페이스 기능

- JpaRepository 인터페이스로 CRUD 기능을 공통으로 처리
- 제네릭은 <엔티티, 식별자>로 설정

```
public interface MemberRepository extends JpaRepository<Member, Long> {
}
```


스프링 데이터
〈〈Interface〉〉
Repository
〈〈Interface〉〉
CrudRepository
save(S) : S
findOne(ID) : T
exists(ID) : boolean
count() : long
delete(T)
...
〈〈Interface〉〉
PagingAndSortingRepository
findAll(Sort) : Iterable〈T〉
findAll(Pageable) : Page〈T〉
스프링 데이터 JPA
〈〈Interface〉〉
JpaRepository
findAll() : List〈T〉
findAll(Sort) : List〈T〉
findAll(Iterable〈ID〉) : List〈T〉
save(Iterable〈S〉) : List〈S〉
flush()
saveAndFlush(T) : T
deleteInBatch(Iterable〈T〉)
deleteAllInBatch()
getOne(ID) : T

쿼리 메서드 기능

- 메서드 이름으로 쿼리 생성
- 메서드 이름으로 JPA NamedQuery 호출
- @Query 어노테이션을 사용해서 리파지토리 인터페이스에 쿼리 직접 정의

메서드 이름으로 쿼리 생성

- 메서드 이름 규칙으로 JPQL 생성

- http://docs.spring.io/spring-data/jpa/docs/1.8.0.RELEASE/reference/html/#jpa.query-methods.query-creation

```
public interface MemberRepository extends Repository<Member, Long> {
    List<Member> findByEmailAndName(String email, String name);
}
```

실행된 JPQL

```
select m from Member m where m.email = ?1 and m.name = ?2
```

@Query, 리파지토리 메서드에 쿼리 정의하기

```
public interface MemberRepository extends JpaRepository<Member, Long> {

    @Query("select m from Member m where m.username = ?1")
    Member findByUsername(String username);
}
```

이름기반 파라미터 바인딩

```
    @Query("select m from Member m where m.username = :name")
    Member findByUsername(@Param("name") String username);
```

반환 타입

```
List<Member> findByName(String name); //컬렉션

Member findByEmail(String email); //단건
```

페이징과 정렬

- org.springframework.data.domain.Sort: 정렬 기능
- org.springframework.data.domain.Pageable: 페이징 기능 (내부에 Sort 포함)

페이징과 정렬

```
//count 쿼리 사용
Page<Member> findByName(String name, Pageable pageable);

//count 쿼리 사용 안함
List<Member> findByName(String name, Pageable pageable);

List<Member> findByName(String name, Sort sort);
```

```java
public interface Page<T> extends Iterable<T> {

    int getNumber();                //현재 페이지
    int getSize();                  //페이지 크기
    int getTotalPages();            //전체 페이지 수
    int getNumberOfElements();      //현재 페이지에 나올 데이터 수
    long getTotalElements();        //전체 데이터 수
    boolean hasPreviousPage();      //이전 페이지 여부
    boolean isFirstPage();          //현재 페이지가 첫 페이지 인지 여부
    boolean hasNextPage();          //다음 페이지 여부
    boolean isLastPage();           //현재 페이지가 마지막 페이지 인지 여부
    Pageable nextPageable();        //다음 페이지 객체, 다음 페이지가 없으면 null
    Pageable previousPageable();//다음 페이지 객체, 이전 페이지가 없으면 null
    List<T> getContent();           //조회된 데이터
    boolean hasContent();           //조회된 데이터 존재 여부
    Sort getSort();                 //정렬 정보
}
```

Web 확장 기능

- @EnableSpringDataWebSupport를 등록해야함
- 페이징과 정렬 기능
- 도메인 클래스 컨버터 기능

Web 페이징과 정렬 기능

- page: 현재 페이지, 0부터 시작
- size: 한 페이지에 노출할 데이터 건수
- sort: 정렬 조건을 정의한다. 예) 정렬 속성,정렬 속성…(ASC | DESC), 정렬 방향을 변경하고 싶으면 sort 파라미터를 추가하면 된다.

/members?page=0&size=20&sort=name,desc

```
@RequestMapping(value = "/members", method = RequestMethod.GET)
public String list(Pageable pageable, Model model) {}
```

도메인 클래스 컨버터 기능

- 컨트롤러에서 식별자로 도메인 클래스 찾음

```
@RequestMapping("/members/{memberId}")
Member member(@PathVariable("memberId") Member member) {
    return member;
}
```

QueryDSL 지원

- org.springframework.data.querydsl.QueryDslPredicateExecutor 상속

```
public interface QueryDslPredicateExecutor<T> {

    T findOne(Predicate predicate);
    Iterable<T> findAll(Predicate predicate);
    Iterable<T> findAll(Predicate predicate, OrderSpecifier<?>... orders);
    Page<T> findAll(Predicate predicate, Pageable pageable);
    long count(Predicate predicate);
}
```

```
repository.findAll(QMember.member.name.eq(name), pageable)
```

실습 ex10-springdatajpa

- Repository에 스프링 데이터 JPA 적용
- @Query 적용
- QueryDSL 적용

실무 경험

- 테이블 중심에서 객체 중심으로 개발 패러다임의 변화
- 유연한 데이터베이스 변경의 장점과 테스트
  - Junit 통합 테스트시에 H2 DB 메모리 모드
  - 로컬 PC에는 H2 DB 서버 모드로 실행
  - 개발 운영은 MySQL, Oracle
- 데이터베이스 변경 경험(로컬은 H2DB, 운영은 오라클, MySQL 중간에 바뀐 적도 있다.)
- 테스트, 통합 테스트시에 CRUD는 믿고 간다.

실무 경험

- 빠른 오류 발견
- 컴파일 시점
- 늦어도 애플리케이션 로딩 시점
- (최소한 쿼리 문법 실수는 오류는 거의 발생하지 않는다.)
- 대부분 비즈니스 오류

실무 경험

- 성능
  - 즉시 로딩: 쿼리가 튐 -> 지연 로딩을 변경
  - N+1 문제 -> 대부분 페치 조인으로 해결
  - 내부 파서 문제: 2000줄 짜리 동적 쿼리 생성 1초
    - 정적 쿼리로 변경(하이버네이트 파싱된 결과 재사용)

기대 사항

- 하이버네이트 5
- 내부 파서 변경
  - 파싱 속도 증가
  - from절의 서브 쿼리
  - 세타 조인 시 **외부 조인 지원(저의 바람)**

감사합니다.