6-2. 양방향 매핑 시 가장 많이 하는 실수

연관관계의 주인에 값을 입력하지 않는 경우...

깃헙 커밋 코드를 참고하자.

양방향 매핑시 순수한 객체 관계를 고려

```
package hellojpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.List;

public class JpaMain {
  public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello");

    EntityManager em = emf.createEntityManager();

    EntityTransaction tx = em.getTransaction();
    tx.begin();
    try {
      Team team = new Team();
      team.setName("TeamA");
      em.persist(team);

      Member member = new Member();
      member.setName("member1");
      member.setTeam(team); //**
      em.persist(member);

      team.getMembers().add(member); // 이걸 안넣어주면 객체지향적인 코드가 아닌거같다? //
**

      em.flush();
```

```
        em.clear();

        Team findTeam = em.find(Team.class, team.getId()); //1차캐시에서 가져온 정보..
(만약 flush와 clear가 없다면!)
        List<Member> members = findTeam.getMembers(); // 지연 로딩!!

        System.out.println("==============");
        for (Member m : members) {
        System.out.println("m = " + m.getName());
}
System.out.println("==============");



        tx.commit();
    } catch (Exception e) {
        tx.rollback();
    } finally {
        em.close();
    }

    emf.close();

  }

}
```

위 처럼 코드를 짜도 멤버의 이름이 나온다!!
왜냐하면, 지연로딩!
팀에서 겟멤버스를 하기 전에 `Team findTeam = em.find(Team.class, team.getId());` 에서 팀을 찾고 그다음 겟멤버스에서 멤버들을 가져옴.
지금 외래키가 다 세팅이 되어있어서 그냥 값이 조회가 됨.

이렇게 하면 약간 객체지향적이지 않다?
위의 객체지향? 표시를 한 코드를 안 넣었을 때의 문제점!!
=> **두 방향 다 값 세팅은 해주되, 주인 객체에 매핑하지 않기는 금물.**

- **만약 flush와 clear를 하지 않았다면, 또** `team.getMembers().add(member` **도 하지 않았다면** 팀과 멤버

뿐만 아니라 설정한 연관관계까지 persist로 인하여 1차 캐시에 들어가있다. 따라서 팀은 그대로 영속성 컨텍스트에 들어가있다. 팀은 컬렉션이 아무것도 없고, 메모리에서도 아무것도 안가져온 상태. 현재 findTeam은 순수한 객체 상태(매핑된게없는상태)인데 그걸 그대로 가져오면 컬렉션도 아무것도 없고 근데 1차캐시에서 로딩은 되니까 아무것도 불러오지 못함.

➜ 다시 한번 정리..

➜ `team.getMembers().add(member);` 와 `em.flush()` , `em.clear()` 를 하지 않으면 팀과 멤버는 persist로 1차캐시, 즉 영속성 컨테스트에만 들어가있고 DB에는 아직 쿼리가 들어가지않아 findTeam에서 아무것도 찾지 못하게 된다. 만약 flush와 clear를 하게 되면 1차 캐시에는 아무것도 없어서 디비까지 가서 찾아오기 때문에 for문에서 출력이 되지만 그렇지 않으면 `team.getMembers().add(member)` 가 없다면 for문에서 아무것도 찾지 못할 것이다.

```
Hibernate:
    call next value for hibernate_sequence
Hibernate:
    call next value for hibernate_sequence
==============
==============
Hibernate:
    /* insert hellojpa.Team
        */ insert
        into
            Team
            (name, id)
        values
            (?, ?)
Hibernate:
    /* insert hellojpa.Member
        */ insert
        into
            Member
            (USERNAME, TEAM_ID, id)
        values
            (?, ?, ?)
```

- 테스트 케이스 작성할 때 jpa 없이도 동작하게끔 순수 자바 코드로만 하는 경우가 많은데 양쪽에 값을 세팅하지 않으면 그게 동작하지 않는다.
- // 표시된 코드를 빼먹을수도있기때문에 아예 도메인 차원에서, 즉 setTeam에서 `team.getMembers().add(this)` (연관관계 편의 메소드)를 써준다. 이 때 set은 잘 쓰지 않고 로직으로 쓸 때는 set같은게 아니라 change같이 좀 특별한 것이라는 것을 표시한다. 혹은 `team.addMember(member)` 로 팀->멤버 방향으로 매핑할 수도 있다.

- 만약 연관관계 편의 메소드가 두 방향 다 있다면 좀 문제가 있을 수 있으니 하나만 쓰자.
- 양방향 매핑시 무한루프 조심!
  - 양방향 매핑 시 toString과 같은 조심!! 롬복에서 toString 만들려면 최대한 쓰지말기

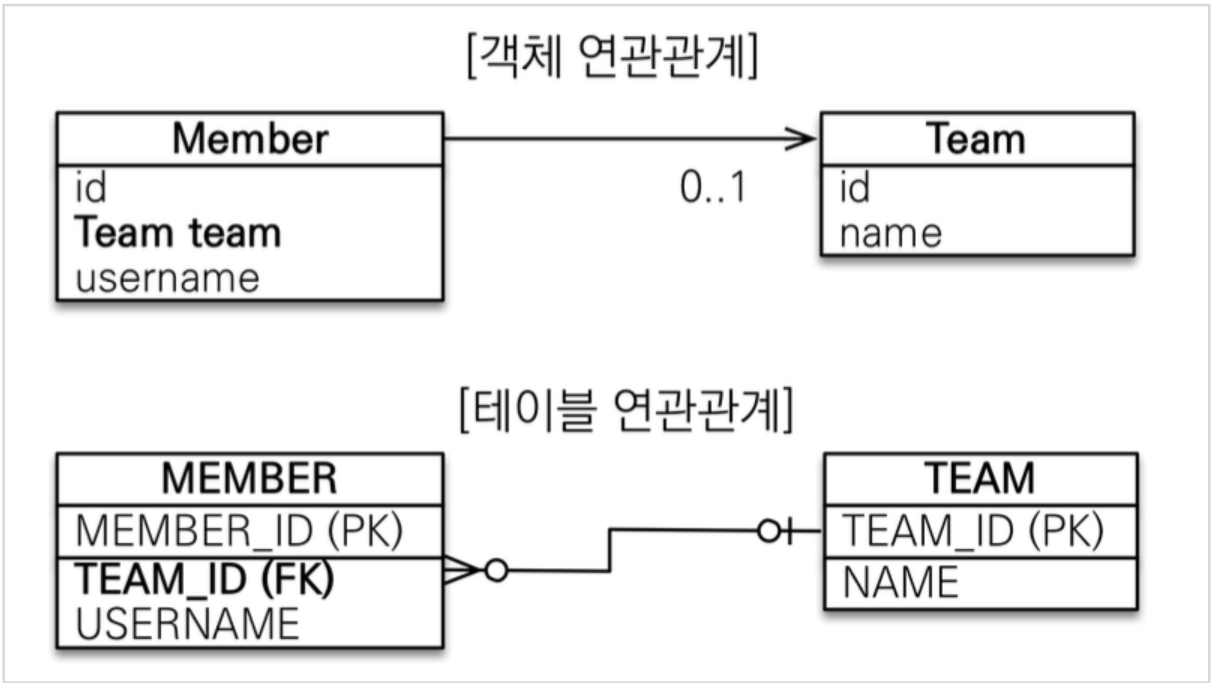
```
        members0_.TEAM_ID=?
Exception in thread "main" java.lang.StackOverflowError Create breakpoint
    at java.base/java.lang.Long.toString(Long.java:490)
    at java.base/java.lang.Long.toString(Long.java:1416)
    at java.base/java.lang.String.valueOf(String.java:4218)
    at java.base/java.lang.StringBuilder.append(StringBuilder.java:173
    at hellojpa.Member.toString(Member.java:51)
    at java.base/java.lang.String.valueOf(String.java:4218)
```

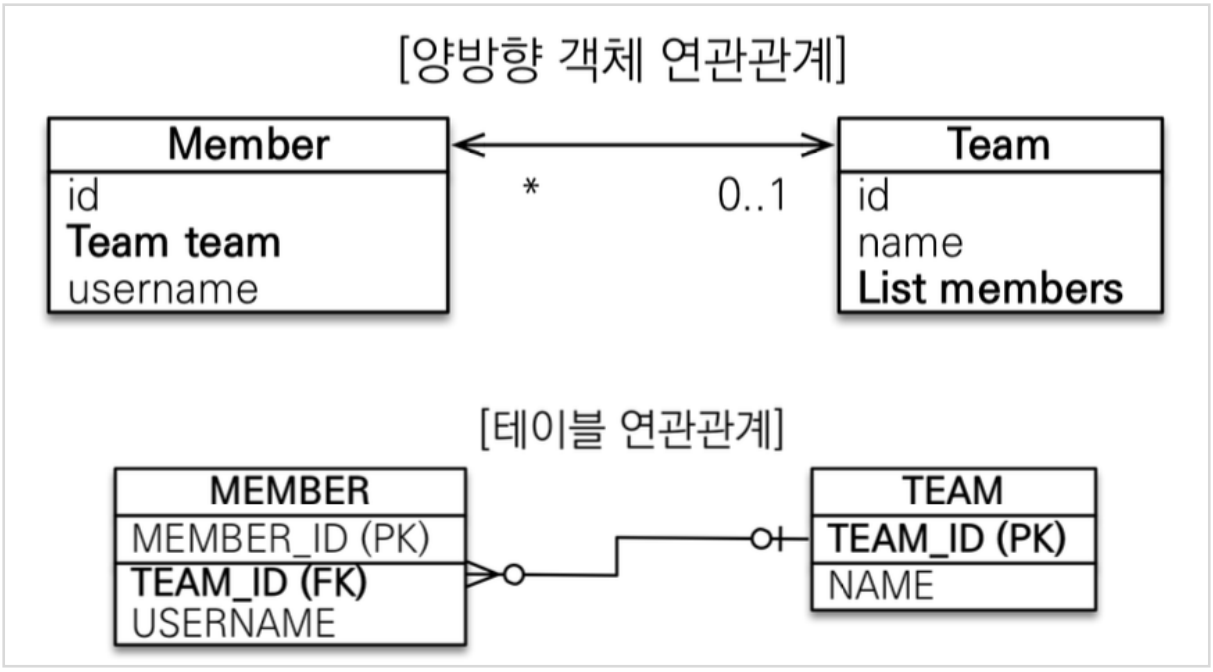
  - JSON 생성 라이브러리 - 컨트롤러에서 엔티티 자체를 반환하지말자!
    - 엔티티를 변경하는 순간 API 스펙이 바뀌어버림 => DTO(단순히값만들어있는것)으로 변환해서 반환하는것이낫다.
    - 무한루프..

양방향 매핑 정리

- 단방향 매핑만으로도 연관관계 매핑은 완료!!!!
  - **처음엔 단방향 매핑으로 설계를 끝낸다!!**

    ➡ 테이블 설계를 어느정도 머리에 그리면서 객체 설계를 그림. 이 때 테이블 차원에서 외래키가 어디에 존재할 지 나옴. 그러면 OneToMany나 ManyToOne으로..
  - 양방향 매핑은 단방향 매핑한 것의 반대 방향으로 **조회 기능이 추가된 것 뿐이다!!!**
  - 단방향 매핑을 잘 하면 양방향은 필요할 때 추가해도 됨!!





➡ 위 그림처럼 단방향에서 양방향으로 바뀌었다고 테이블 연관관계는 건들지 않기 때문에 그냥 추가만 하는 방향이 좋다.

=> 기본적으로는 단방향 매핑으로는 다 끝낸다. (many 에다가 연관관계 매핑 해주고)

#JPA