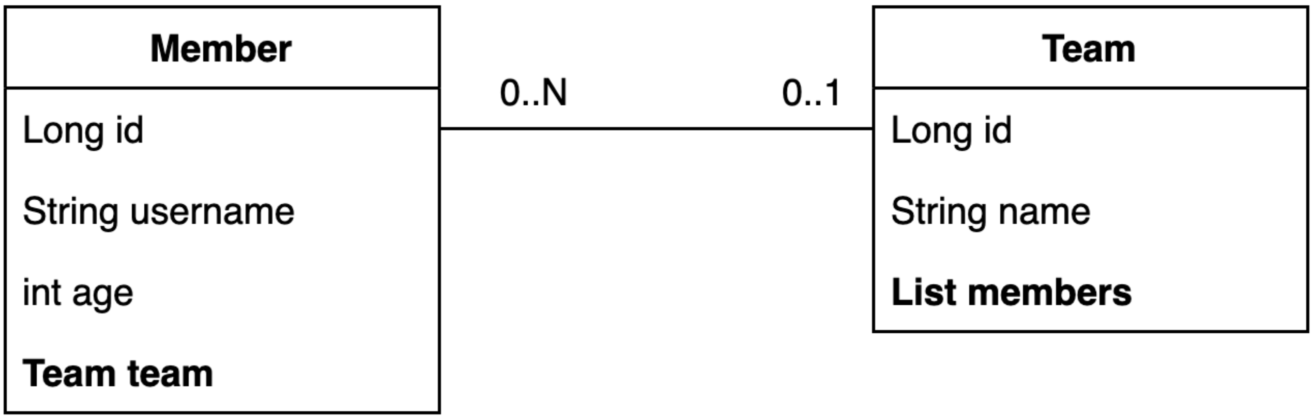
2. 예제 도메인 모델

#1.인강/jpa활용편/datajpa/강의

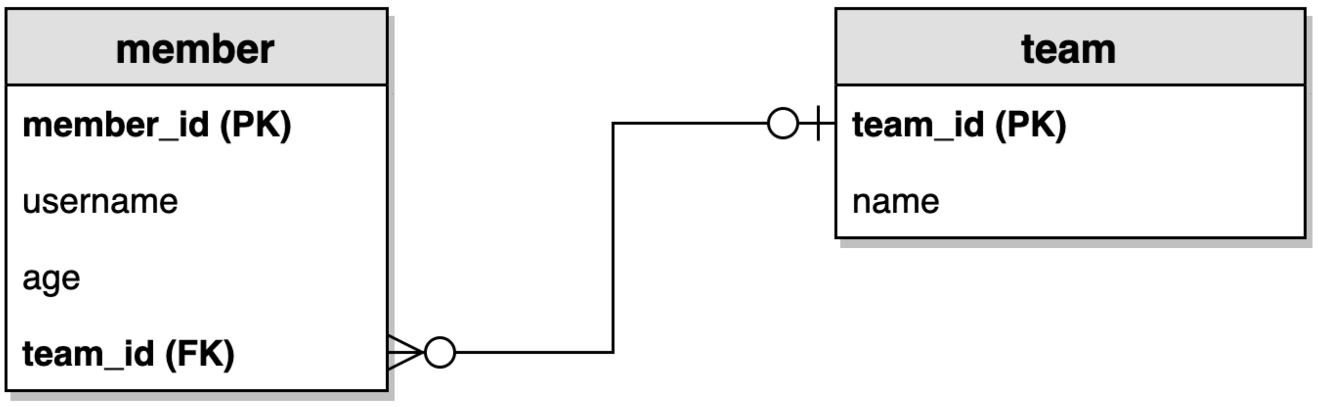
- /예제 도메인 모델과 동작확인

예제 도메인 모델과 동작확인

엔티티 클래스



ERD



Member 엔티티

```
package study.datajpa.entity;

import lombok.AccessLevel;
```

```java
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Getter @Setter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@ToString(of = {"id", "username", "age"})
public class Member {

    @Id
    @GeneratedValue
    @Column(name = "member_id")
    private Long id;
    private String username;
    private int age;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "team_id")
    private Team team;

    public Member(String username) {
        this(username, 0);
    }

    public Member(String username, int age) {
        this(username, age, null);
    }

    public Member(String username, int age, Team team) {
        this.username = username;
        this.age = age;
        if (team != null) {
            changeTeam(team);
        }
    }

    public void changeTeam(Team team) {
        this.team = team;
        team.getMembers().add(this);
    }
}
```

- 롬복 설명
    - @Setter: 실무에서 가급적 Setter는 사용하지 않기
    - @NoArgsConstructor AccessLevel.PROTECTED: 기본 생성자 막고 싶은데, JPA 스팩상 PROTECTED로 열어두어야 함
    - @ToString은 가급적 내부 필드만(연관관계 없는 필드만)
- `changeTeam()` 으로 양방향 연관관계 한번에 처리(연관관계 편의 메소드)

Team 엔티티

```java
package study.datajpa.entity;

import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
@Getter @Setter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@ToString(of = {"id", "name"})
public class Team {

    @Id @GeneratedValue
    @Column(name = "team_id")
    private Long id;
    private String name;

    @OneToMany(mappedBy = "team")
    List<Member> members = new ArrayList<>();

    public Team(String name) {
        this.name = name;
    }
}
```

- Member와 Team은 양방향 연관관계, `Member.team`이 연관관계의 주인, `Team.members`는 연관관계의 주인이 아님, 따라서 `Member.team`이 데이터베이스 외래키 값을 변경, 반대편은 읽기만 가능

데이터 확인 테스트

```
package study.datajpa.entity;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.annotation.Rollback;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;

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

@SpringBootTest
public class MemberTest {

    @PersistenceContext
    EntityManager em;

    @Test
    @Transactional
    @Rollback(false)
    public void testEntity() {
        Team teamA = new Team("teamA");
        Team teamB = new Team("teamB");
        em.persist(teamA);
        em.persist(teamB);

        Member member1 = new Member("member1", 10, teamA);
        Member member2 = new Member("member2", 20, teamA);
        Member member3 = new Member("member3", 30, teamB);
        Member member4 = new Member("member4", 40, teamB);

        em.persist(member1);
        em.persist(member2);
        em.persist(member3);
        em.persist(member4);
        //초기화
        em.flush();
        em.clear();

        //확인
        List<Member> members = em.createQuery("select m from Member m",
Member.class)
                .getResultList();

        for (Member member : members) {
            System.out.println("member=" + member);
            System.out.println("-> member.team=" + member.getTeam());
        }
    }
}
```

- 가급적 순수 JPA로 동작 확인 (뒤에서 변경)
- db 테이블 결과 확인
- 지연 로딩 동작 확인