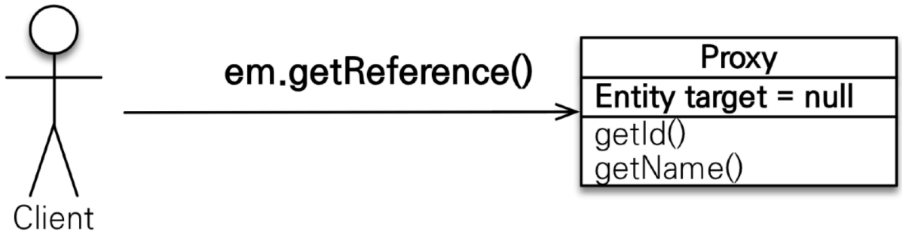
JPA 프록시와 연관관계 관리

Why?

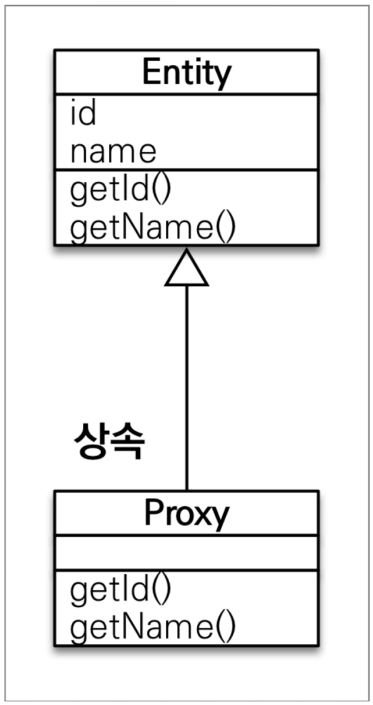
연관관계 데이터 가져올 때 매번 join 결과를 가져와야할까?
➜ 불필요한 리소스를 사용하게 된다.

프록시 기초

- em.find() vs em.getReference()
  - em.find(): 데이터베이스를 통해서 실제 엔티티 객체 조회
  - em.getReference(): 데이터베이스 조회를 미루는 가짜(프록시) 엔티티 객체 조회
    - ex) Member findMember = em.getReference(Member.class, member.getId());
      findMember.getId(); ➜ member.getId() 는 이미 있기 때문에 DB에 요청하지 않는다.
      findMember.getUsername(); ➜ member.getUsername 은 데이터가 없기 때문에 DB에 요청한다.



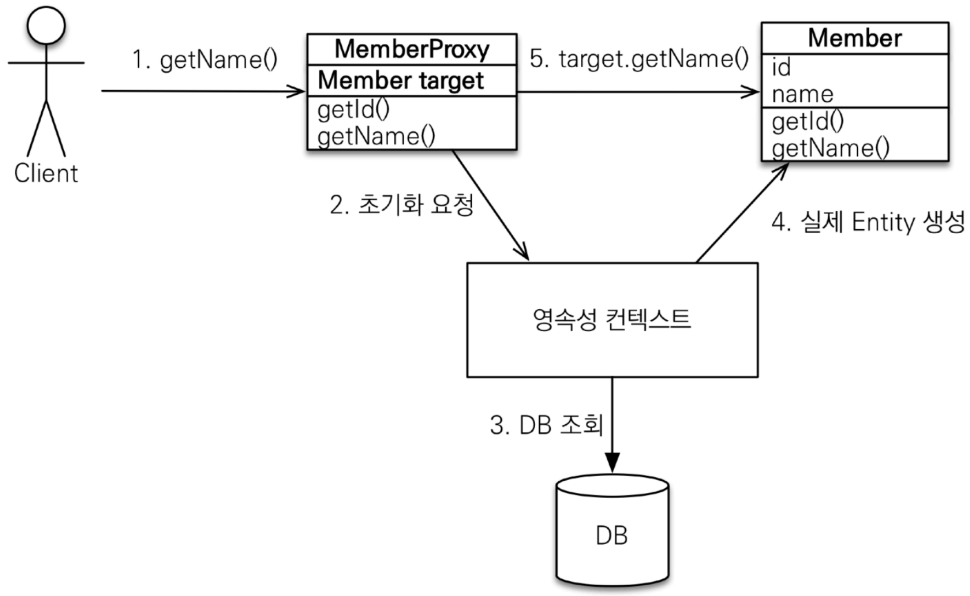
- 실제 클래스를 상속 받아서 만들어짐
- 실제 클래스와 겉 모양이 같다.
- 사용하는 입장에서는 진짜 객체인지 프록시 객체인지 구분하지 않고 사용하면 됨(이론상)



- 프록시 객체는 실제 객체의 참조(target)를 보관
- 프록시 객체를 호출하면 프록시 객체는 실제 객체의 메소드 호출

프록시 객체의 초기화

```
Member member = em.getReference(Member.class, "id1");
member.getName();
```



- **프록시 객체는 처음사용할때 한번만 초기화**
- 프록시 객체를 초기화할 때, 프록시 객체가 실제 엔티티로 바뀌는것은 아님,

- 초기화되면 프록시 객체를 통해서 실제 엔티티에 접근 가능
- **프록시 객체는 원본 엔티티를 상속받음, 따라서 타입 체크시 주의해야함**

  (== 비교 실패, 대신 instance of 사용)
- 영속성 컨텍스트에 찾는 엔티티가 이미 있으면 em.**getReference()**를 호출해도 실제 엔티티 반환

```
Member m1 = em.find(Member.class, member1.getId());
System.out.println("m1 = " + m1.getClass());

Member reference = em.getReference(Member.class, member1.getId());
System.out.println("reference = " + reference.getClass());

System.out.println("a == a: " + (m1 == reference));
```

  - 한 트랜젝션 안에 처리를 보장해주기 위해 entity 반환
  - 프록시로 반환해서 처리하는 이점이 없기 때문에

```
Member refMember = em.getReference(Member.class, member1.getId());
System.out.println("refMember = " + refMember.getClass()); //Proxy

Member findMember = em.find(Member.class, member1.getId());
System.out.println("findMember = " + findMember.getClass()); //Member

System.out.println("refMember == findMember: " + (refMember == findMember));
```

    - refMember == findMember 결과 는 true
- 영속성 컨텍스트의 도움을 받을 수 없는 준영속 상태일 때, 프록시를 초기화하면 문제 발생
  - (하이버네이트는 org.hibernate.LazyInitializationException 예외를 터트림)

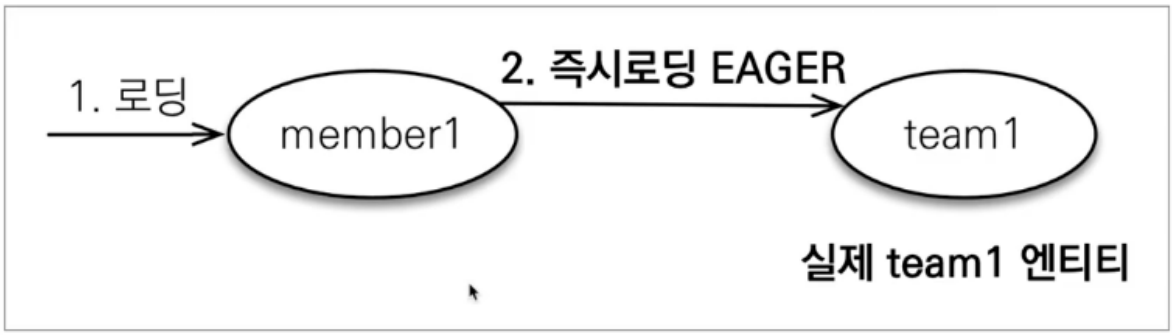
프록시 확인

- **프록시 인스턴스의 초기화 여부 확인**
  - PersistenceUnitUtil.isLoaded(Object entity)
- **프록시 클래스 확인 방법**
  - entity.getClass().getName() 출력(..javasist.. or HibernateProxy...)
- **프록시 강제 초기화**
  - org.hibernate.Hibernate.initialize(entity);

즉시 로딩과 지연로딩

- **지연로딩 (fetch = FetchType.LAZY)**
  - 지연로딩 설정하면 프록시로 조회 해온다.

- team을 실제 사용하는 시점에 프록시 초기화 (DB조회)

- **즉시 로딩 (fetch = FetchType.EAGER)**
  - 

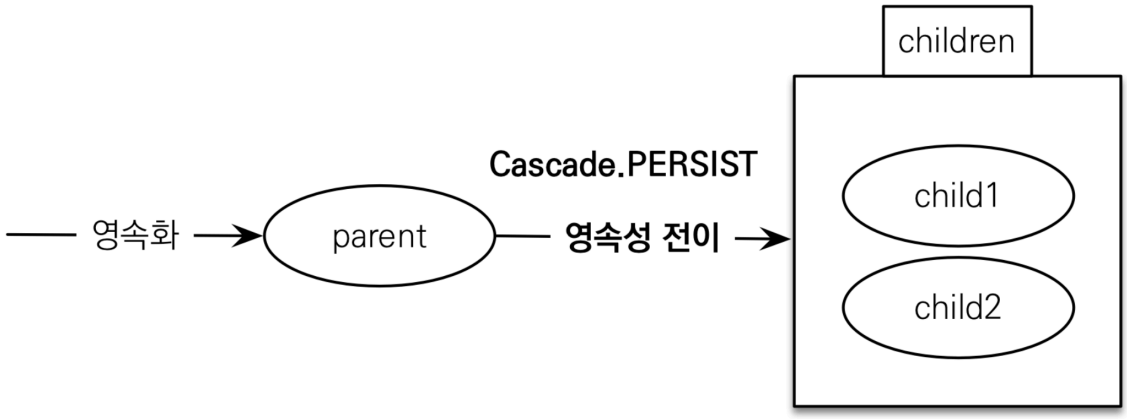
주의 사항

- **가급적 지연 로딩만 사용 (특히 실무에서)**
- **즉시 로딩을 적용하면 예상하지 못한 SQL이 발생**
- **즉시 로딩은 JPQL에서 N+1 문제를 일으킨다**
  - JPQL 은 쿼리로 조회를 하는데 조회해서 가져온 값에 jpa 에서 즉시로딩으로 되어 있으면 jpa에서 쿼리를 또 조회하게된다. 결과 N 마다 + 1 만큼 조회한다.
  - 해결방법
    - 모든 걸 지연로딩으로 변경한다.
    - fetch join
    - 엔티티 그래프 기능을 사용해라
- **@ManyToOne, @OneToOne 은 기본이 즉시 로딩 → LAZY로 설정**

CASCADE (영속성 전이)

영속성 전이: 저장

```
@OneToMany(mappedBy="parent", cascade=CascadeType.PERSIST)
```



- 영속성 전이는 연관관계를 매핑하는 것과 아무 관련이 없음
- 엔티티를 영속화할 때 연관된 엔티티도 함께 영속화하는 **편리함** 을 제공할 뿐

- CASCADE 종류
  - **ALL** : **모두 적용**
  - **PERSIST**: **영속** 저장할 때만 쓸땐
  - **REMOVE** : **삭제**
  - MERGE
  - REFRESH
  - DETACH
- 부모 자식간에서만 사용하는 경우에만 사용.
- 만약 다른 곳에서 자식을 사용해야한다면 안쓰는게 낫다.
- 라이프사이클이 똑같을 때 사용
- 단일 소유자일 경우 사용

고아 객체

- 고아 객체 제거: 부모 엔티티와 연관관계가 끊어진 자식 엔티티 를 자동으로 삭제
- **orphanRemoval = true**

- 참조가 제거된 엔티티는 다른 곳에서 참조하지 않는 고아 객체로 보고 삭제하는 기능
- 참조하는 곳이 하나일 경우일 때 사용
- 특정 엔티티가 개인 소유할 때 사용
- @OneToOne, @OneToMany만 가능
- 참고: 개념적으로 부모를 제거하면 자식은 고아가 된다. 따라서 고 아 객체 제거 기능을 활성화 하면, 부모를 제거할 때 자식도 함께 제거된다. 이것은 CascadeType.REMOVE처럼 동작한다.

영속성 전이 + 고아 객체, 생명주기

- **CascadeType.ALL + orphanRemoval=true**
- 스스로 생명주기를 관리하는 엔티티는 em.persist()로 영속화, em.remove()로 제거
- 두 옵션을 모두 활성화 하면 부모 엔티티를 통해서 자식의 생명 주기를 관리할 수 있음
- 도메인 주도 설계(DDD)의 Aggregate Root개념을 구현할 때 유용

#dev/jpa/tutorial