멋쟁이 사자처럼
백엔드 세션 2주차

1. JPA를 사용하지 않는다면?

```
# 회원 객체
public class Memer {

  private String memberId;
    private String name;
  ...
    }
```

```
# 회원 DAO(데이터 접근 객체)
public class MemberDAO{
  public Member find(String memberId){...}
    }
```

1. 회원 조회용 SQL 작성.
2. JDBC API 사용해서 SQL 실행.
3. 조회 결과를 Member 객체로 매핑

객체를 데이터베이스에 CRUD하려면
매우 많은 SQL과 JDBC API 코딩 필요

2. 요구 사항이 추가된다면?

회원에 연락처도 함께 저장해 달라는 요구 사항 추가된다면?

1. 등록 코드 변경

회원 객체에 연락처 필드 추가하고, INSERT SQL 수정

2. 조회 코드 변경

회원 조회용 SQL에 연락처 컬럼 추가, 조회 결과를 회원 객체에 추가로 매핑.

3. 수정 코드 변경

연락처가 수정되지 않는 버그 발생 시 MemberDAO를 열어 UPDATE SQL 확인.

4. 연관된 객체(요구사항 추가)

회원은 어떤 한 팀에 필수로 소속된다는 요구사항 추가.

MemberDAO를 열어 확인 후 SQL 수정.

3. SQL의 문제점

1. 데이터 접근 계층을 사용해 SQL을 숨겨도 접근 계층을 열어서 어떤 SQL이 실행되는지 확인해야 함

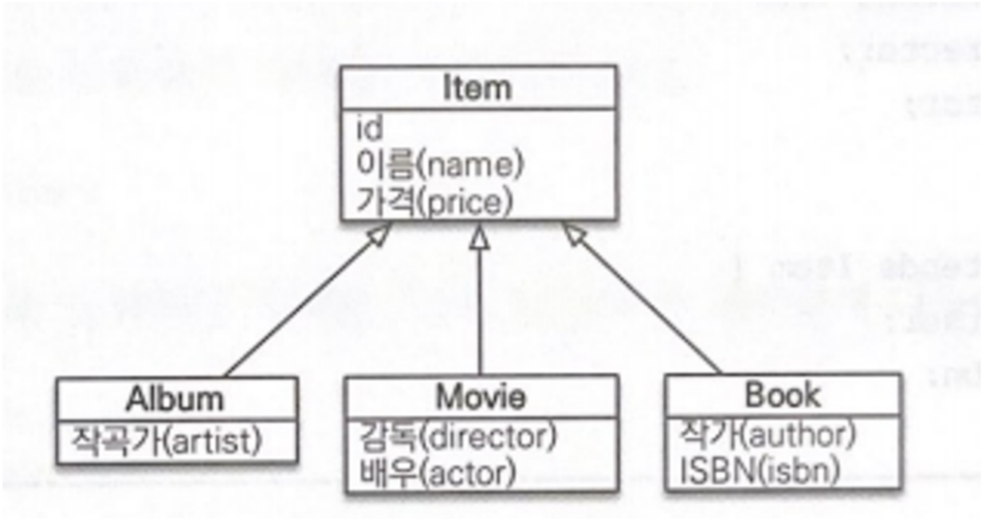
2. SQL에 의존하는 상황에서는 개발자들이 엔티티를 신뢰할 수 없다.

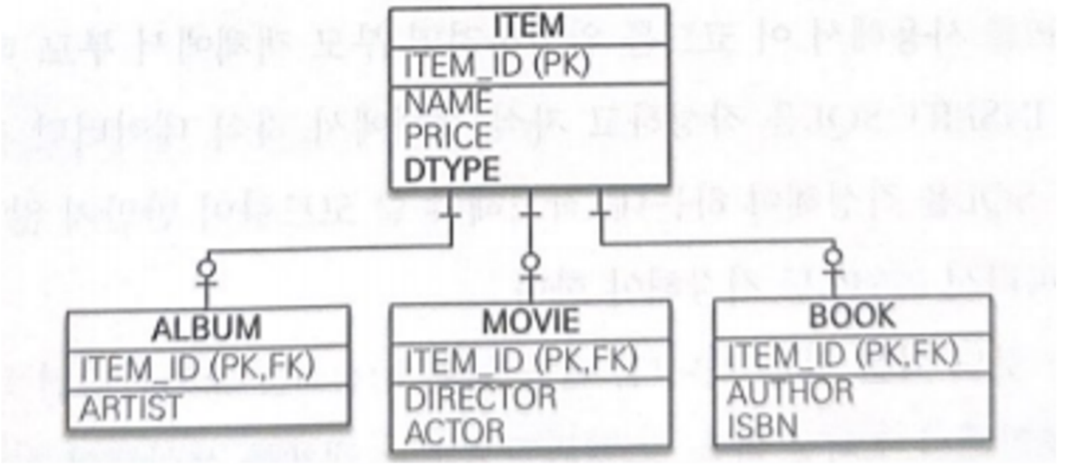
3. 계층분할의 모순
물리적으로는 SQL과 JDBC API를 데이터 접근 계층에 숨겼다.
논리적으로는 엔티티와 아주 강한 의존관계를 가지고 있다.
항상 DAO를 열어서 어떤 SQL이 실행되는지 확인해야 한다.
진정한 의미의 계층분할이 아님.

4. JPA와 문제 해결

JPA를 사용하면 객체를 DB에 저장하고 관리할 때, JPA가 제공하는 API를 사용하면 된다.
JPA는 개발자 대신 SQL을 생성해 DB에 전달한다(CRUD).

5. 패러다임 불일치 문제 : 상속





객체는 상속기능을 가지고 있지만 테이블은 상속기능이 없다.
슈퍼타입 서브타입 관계를 사용하면 유사하게 설계할 수 있지만 정확히 일치하지 않음.

6. 상속 문제 해결

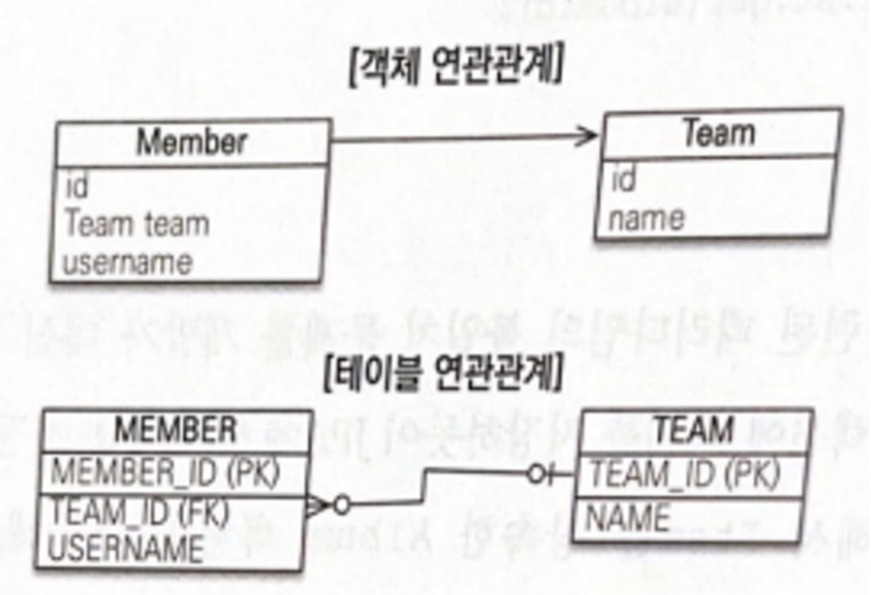
```
# 객체 모델 코드
abstract class Item {
  Long id;
  String name;
    int price;
    }

class Album extends Item {
  String artist;
}
class Movie extends Item {
  String director;
  String actor;
}
class Book extends Item {
  String author;
  String isbn;
}
```

JDBC API를 사용해서 코드를 완성하려면
ITEM용 INSERT SQL, ALBUM용 INSERT SQL 각각 작성.

JPA는?
Item을 상속한 Album 객체를 저장해보자.

1. persist() 메소드 실행
2. JPA는 다음 SQL을 실행해 객체를 두 테이블에 나누어 저장.
INSERT INTO ITEM...
INSERT INTO ALBUM...

6. 패러다임의 불일치 문제 : 연관관계



- 객체

참조를 사용해 다른 객체와 연관관계를 가지고 참조에 접근해 연관된 객체를 조회.

- 테이블

외래키를 사용해 다른 테이블과 연관관계를 가지고 조인을 사용해서 연관된 테이블을 조회.

6. 패러다임의 불일치 문제 : 연관관계

테이블에 맞춰 설계한 객체

```
class Member{

  String id;
    Long teamId;
    String username;
}

class Team{

  Long id;
    String name;
}
```

--------->

참조를 사용하도록 모델링

```
class Member{

  String id;
    Team team;
    String username;

    Team getTeam(){
      return team;
    }
}

class Team{

  Long id;
    String name;
}
```

6. 패러다임의 불일치 문제 : 연관관계

```
member.getId(); //MEMBER_ID PK에 저장
member.getTeam().getId(); //TEAM_ID FK에 저장
member.getUsername(); // USERNAME 컬럼에 저장
```

개발자는 중간에서 변환 역할을 해줘야 한다.

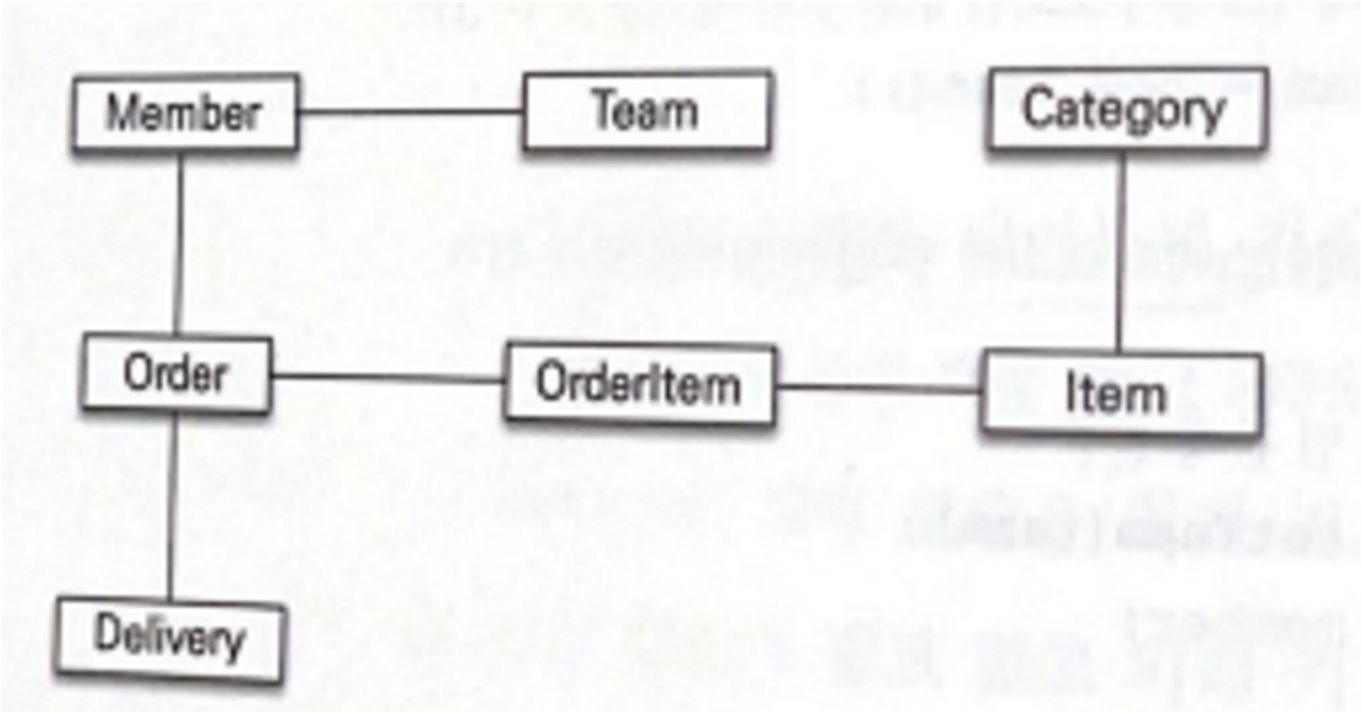
저장의 경우 team 필드를 TEAM_ID 외래 키 값으로 변환해야 한다.
그 후 값을 찾아서 INSERT SQL을 만들어야 한다

7. 연관관계 문제 해결

```
member.setTeam(team); //회원과 팀 연관관계 설정
jpa.persist(member); //회원과 연관관계 함께 저장
```

JPA는 team의 참조를 외래키로 변환해 INSERT SQL을 데이터베이스에 전달
객체를 조회할 때 외래키를 참조로 변환하는 일도 해줌.

8. 객체 그래프 탐색



JPA는 연관된 객체를 사용하는 시점에 SELECT SQL을 실행한다(지연 로딩).

9. 값 비교

```
String memberId = "100";
Member member1 = memberDAO.getMember(memberId);
Member member2 = memberDAO.getMember(memberId);

member1 == member2;
```

TRUE or FALSE?

9. 값 비교

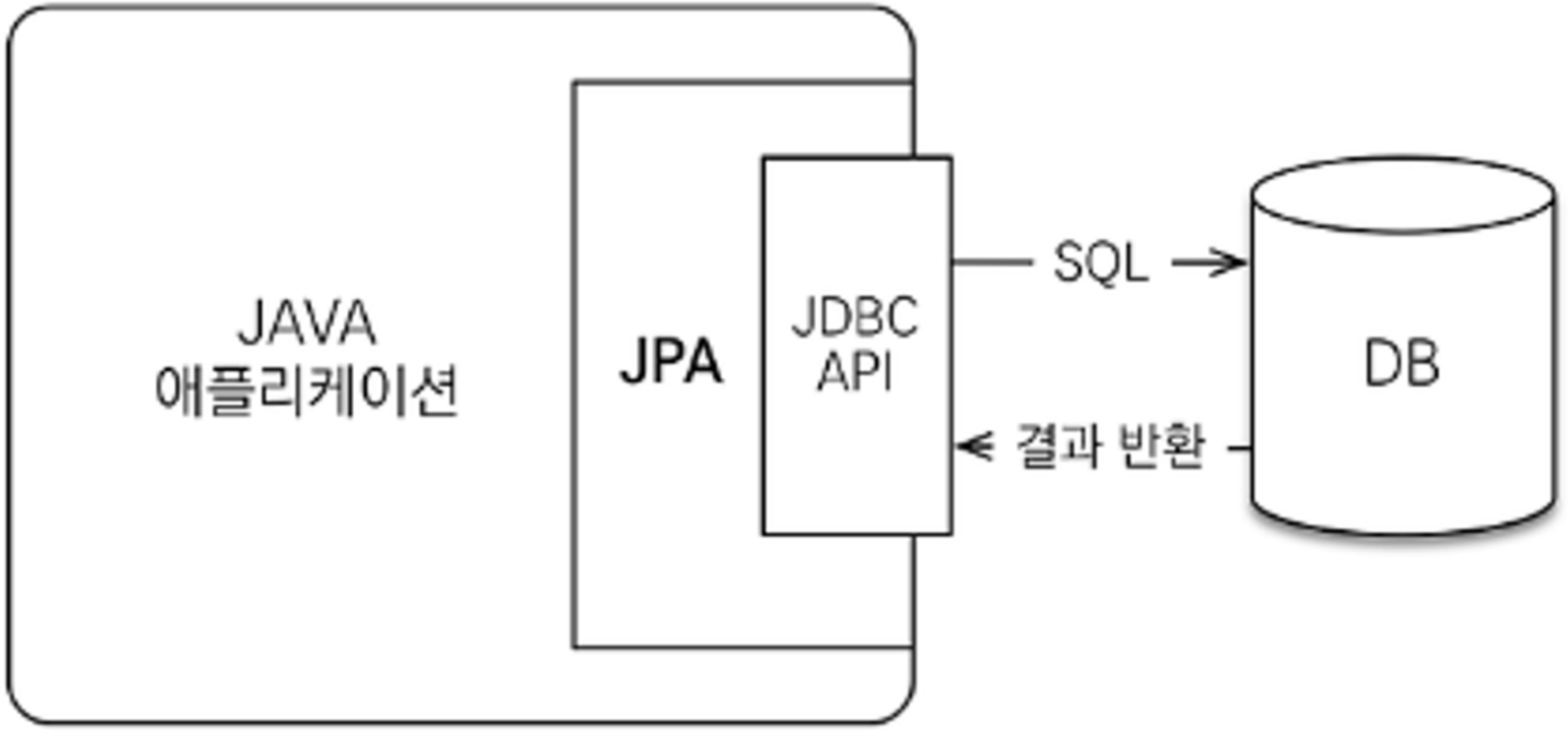
1. 동일성 비교(==)
 객체 인스턴스의 주소 값을 비교한다.
2. 동등성 비교(equals())
객체 내부의 값을 비교한다.

```
String memberId = "100";
Member member1 = jpa.find(Member.class, memberId);
Member member2 = jpa.find(Member.class, memberId);

member1 == member2;
```

데이터베이스는 기본키의 값으로 각 열(row)구분.
객체는 동일성 비교와 동등성 비교의 두 가지 방법.

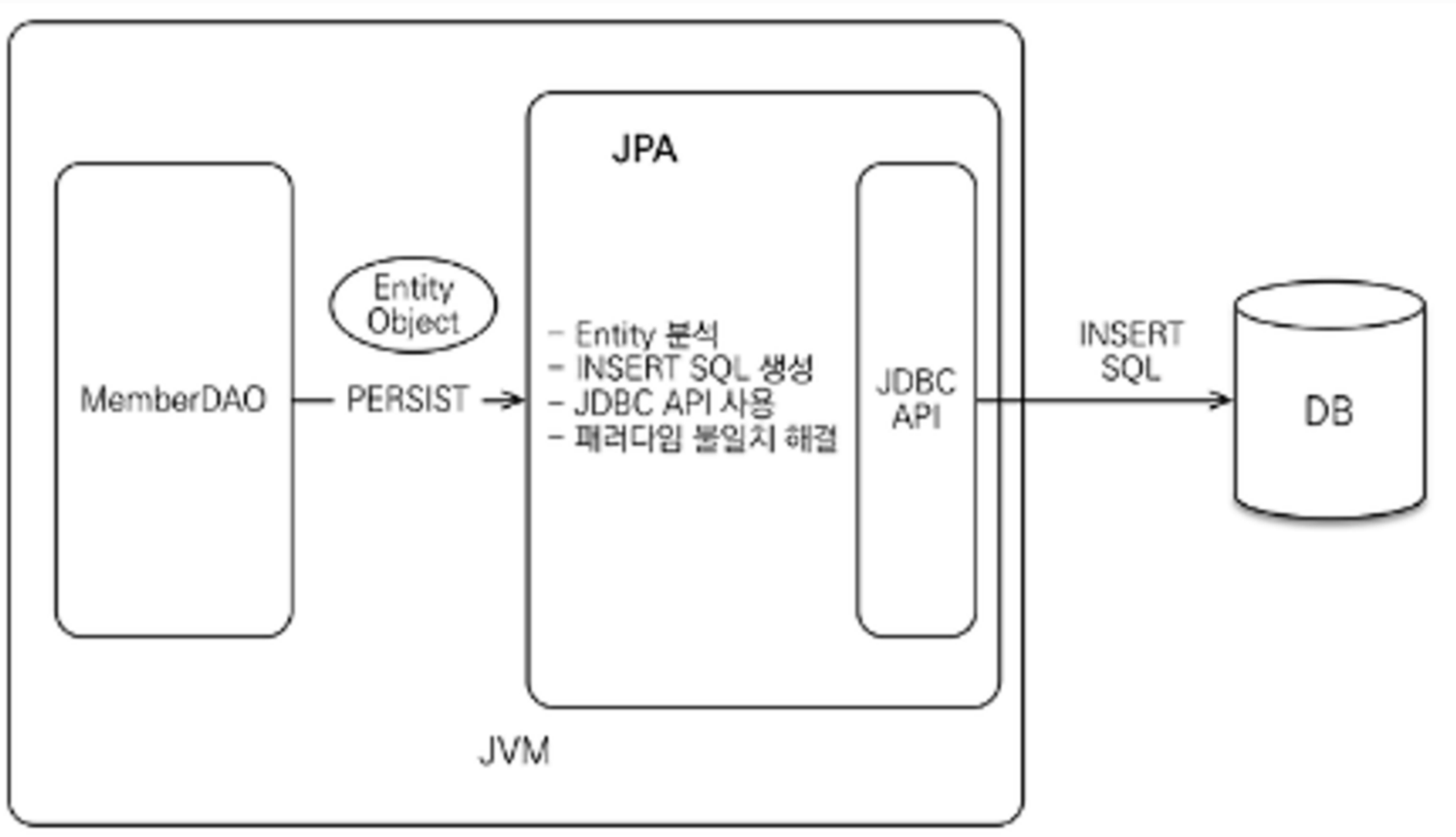
JPA는 같은 트랙잭션일 때 같은 객체가 조회되는 것을 보장한다.

10. JPA란 무엇인가?



JPA(Java Persistence API)는 자바 진영의 ORM기술 표준으로
애플리케이션과 JDBC 사이에서 동작한다.

10. JPA란 무엇인가?



ORM(Obeject-Relational Mapping)은 객체와 관계형 데이터베이스를 매핑한다.
ORM 프레임워크는 객체와 테이블을 매핑해서 패러다임의 불일치 문제를 해결한다

11. JPA의 사용 이유

1. 생산성
CREATE TABEL 같은 DDL 문을 자동으로 생성해줌.

2. 유지보수
SQL을 직접 다루면 엔티티에 필드를 하나만 추가해도 결과 매핑을 위한 JDBC API 코드를 모두 변경해야 한다.
JPA는 이런 과정을 대신 해주므로 유지 보수해야 하는 코드 수가 줄어든다.

3. 패러다임의 불일치 해결

4. 성능