

9. 값 타입

1. 기본값 타입
- ★ 2. 임베디드 타입 (복합 값 타입)
3. 값 타입과 불변 객체
4. 값 타입의 비교
- ★ 5. 값 타입 컬렉션
6. 실전 예제 - 6. 값 타입 매핑

9. 값 타입 - 기본값 타입

<JPA의 데이터 타입 분류>

1. 엔티티 타입

- @Entity로 정의하는 객체
- 데이터가 변해도 식별자로 지속해서 추적 가능
- 예) 회원 엔티티의 키나 나이 값이 변경해도 식별자로 인식 가능

2. 값 타입

- int, Integer, String처럼 단순히 값으로 사용하는 자바 기본 타입이나 객체
- 식별자가 없고 값만 있으므로 변경시 추적 불가
- 예) 숫자 100을 200으로 변경하면 완전히 다른 값으로 대

9. 값 타입 - 기본값 타입

<값 타입 분류>

1. 기본값 타입

- 자바 기본 타입(int, double)
- 래퍼 클래스(Integer, Long)
- String

2. 임베디드 타입(embedded type, 복합 값 타입)

3. 컬렉션 값 타입(collection value type)

9. 값 타입 - 기본값 타입

<1. 기본값 타입>

- 생명주기를 엔티티에 의존

ex) 회원이 삭제하면 이름, 나이 필드도 함께 삭제

- 공유 X

ex) 회원 이름 변경 시 다른 회원의 이름도 함께 변경되면 안됨

- int, double 같은 기본 타입은 절대 공유 X

- 기본 타입은 값을 항상 복사함

```
int a = 10;  
int b = a;  
  
a=20;  
  
System.out.println("a= "+a);  
System.out.println("b= "+b);
```

a= 20

b= 10

9. 값 타입 - 임베디드 타입

<2. 임베디드 타입-복합 값 타입>

- 새로운 값 타입을 직접 정의할 수 있음
- JPA는 임베디드 타입(embedded type)이라 함
- 주로 기본 값 타입을 모아서 만들어서 복합 값 타입이라고도 함
- `int`, `String`과 같은 값 타입

9. 값 타입 - 임베디드 타입

<임베디드 타입 사용법>

- @Embeddable: 값 타입을 정의하는 곳에 표시
- @Embedded: 값 타입을 사용하는 곳에 표시
- 기본 생성자 필수

<임베디드 타입 장점>

- 재사용
- 높은 응집도
- Period.isWork()처럼 해당 값 타입만 사용하는 의미 있는 메소드를 만들 수 있음
- 임베디드 타입을 포함한 모든 값 타입은, 값 타입을 소유한 엔티티에 생명주기를 의존함

```
//근무 기간
@Temporal(TemporalType.DATE)
Date startDate;

@Temporal(TemporalType.DATE)
Date endDate;

//집 주소
private String city;
private String street;
private String zipcode;
```

```
//근무 기간
@Embedded
Period workPeriod;

//집 주소
@Embedded
Address homeAddress;
```

```
@Embeddable
public class Address {
    @Column(name="member_city")
    private String city;
    private String street;
    private String zipcode;
}

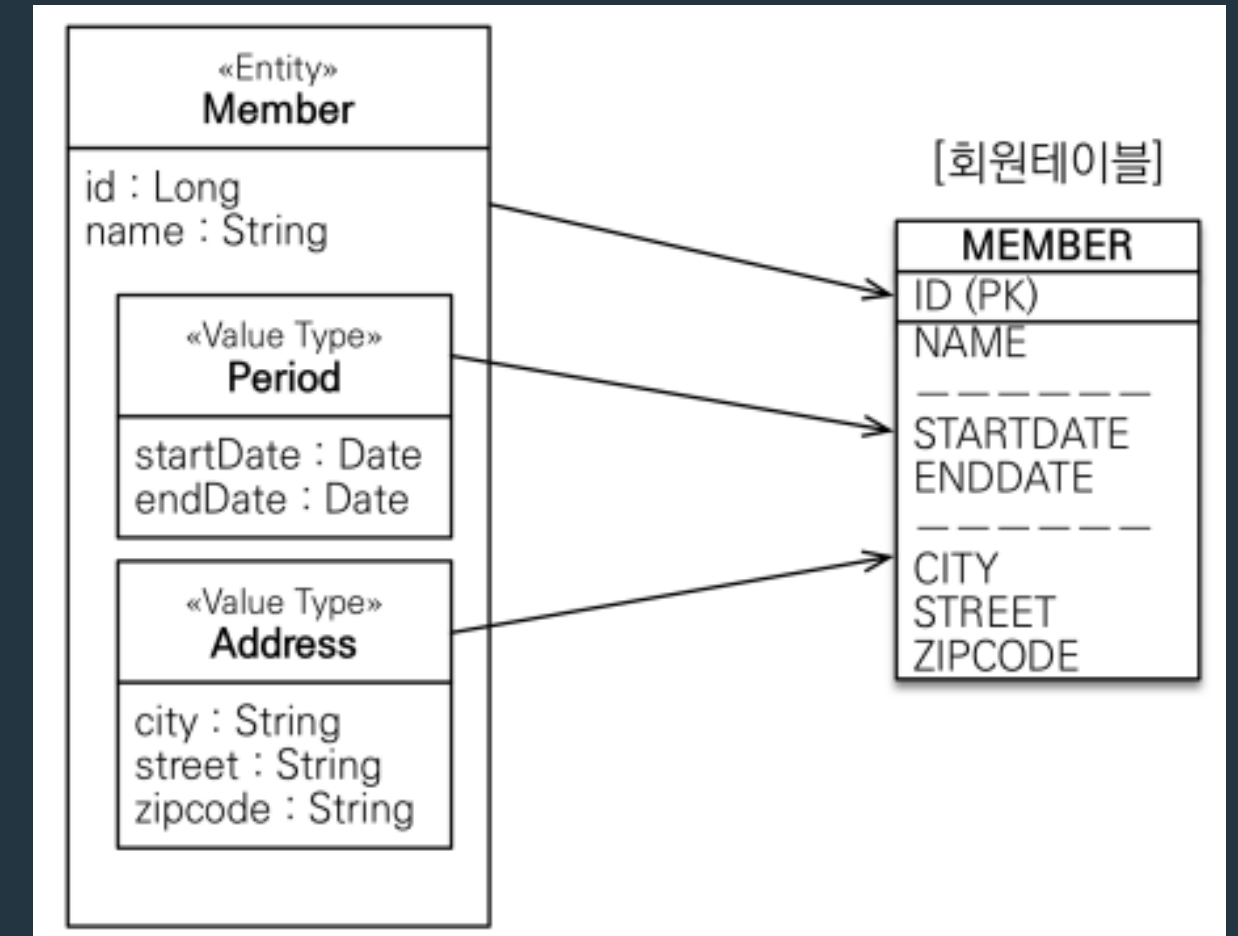
@Embeddable
public class Period {
    @Temporal(TemporalType.DATE)
    Date startDate;
    @Temporal(TemporalType.DATE)
    Date endDate;

    public boolean isWork(Date date){
        //.. 값 타입을 위한 메서드를 정의할 수도 있습니다.
    }
}
```

9. 값 타입 - 임베디드 타입

<임베디드 타입과 테이블 매핑>

- 임베디드 타입은 엔티티의 값일 뿐이다.
- 임베디드 타입을 사용하기 전과 후에 **매핑하는 테이블은 같다.**
- 객체와 테이블을 아주 세밀하게(find-grained) 매핑하는 것이 가능
- 잘 설계한 ORM 애플리케이션은 매핑한 테이블의 수보다 클래스의 수가 더 많음

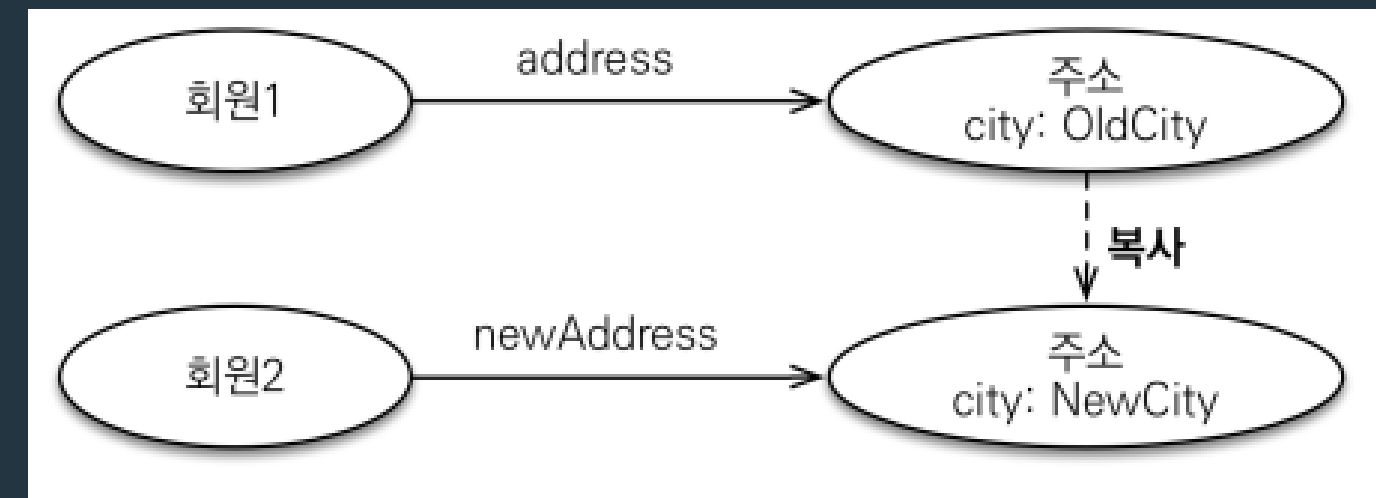
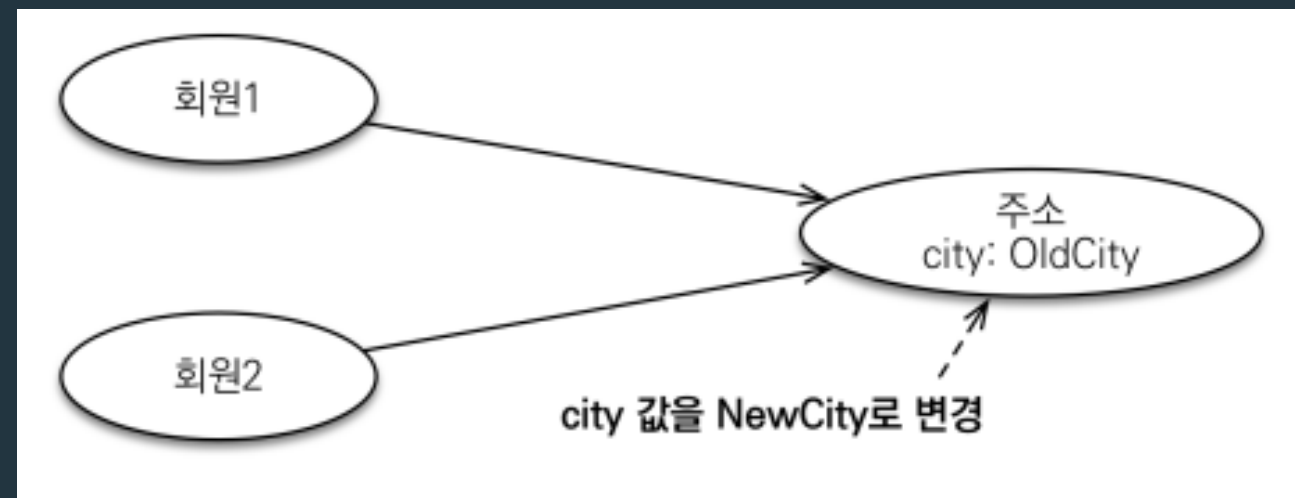


9. 값 타입 - 값 타입과 불변 객체

값 타입은 복잡한 객체 세상을 조금이라도 단순화하려고 만든 개념이다. 따라서 값 타입은 단순하고 안전하게 다룰 수 있어야 한다.

<값 타입 공유 참조>

- 임베디드 타입 같은 값 타입을 여러 엔티티에서 공유하면 위험함
- 부작용(side effect) 발생



9. 값 타입 - 값 타입과 불변 객체

- 항상 값을 복사해서 사용하면 공유 참조로 인해 발생하는 부작용을 피할 수 있다.
- 문제는 임베디드 타입처럼 직접 정의한 값 타입은 자바의 기본타입이 아니라 객체 타입이다.
- 자바 기본 타입에 값을 대입하면 값을 복사한다.
- 객체 타입은 참조 값을 직접 대입하는 것을 막을 방법이 없다.
- 객체의 공유 참조는 피할 수 없다.

기본 타입(primitive type)

```
int a = 10;  
int b = a; //기본 타입은 값을 복사  
b = 4;
```

객체 타입

```
Address a = new Address("Old");  
Address b = a; //객체 타입은 참조를 전달  
b.setCity("New")
```

9. 값 타입 - 값 타입과 불변 객체

<불변 객체>

- 객체 타입을 수정할 수 없게 만들면 부작용을 원천 차단
- 값 타입은 불변 객체(immutable object)로 설계해야함
- 불변 객체: 생성 시점 이후 절대 값을 변경할 수 없는 객체
- 생성자만으로 값을 설정하고 수정자(Setter)를 만들지 않으면 됨
- 참고: Integer, String은 자바가 제공하는 대표적인 불변 객체

9. 값 타입 - 값 타입의 비교

- 값 타입: 인스턴스가 달라도 그 안에 값이 같으면 같은 것으로 봐야 함

<기본 타입>

int a = 10;

int b = 10;

a==b? T

<객체 타입>

Address a = new Address("서울시")

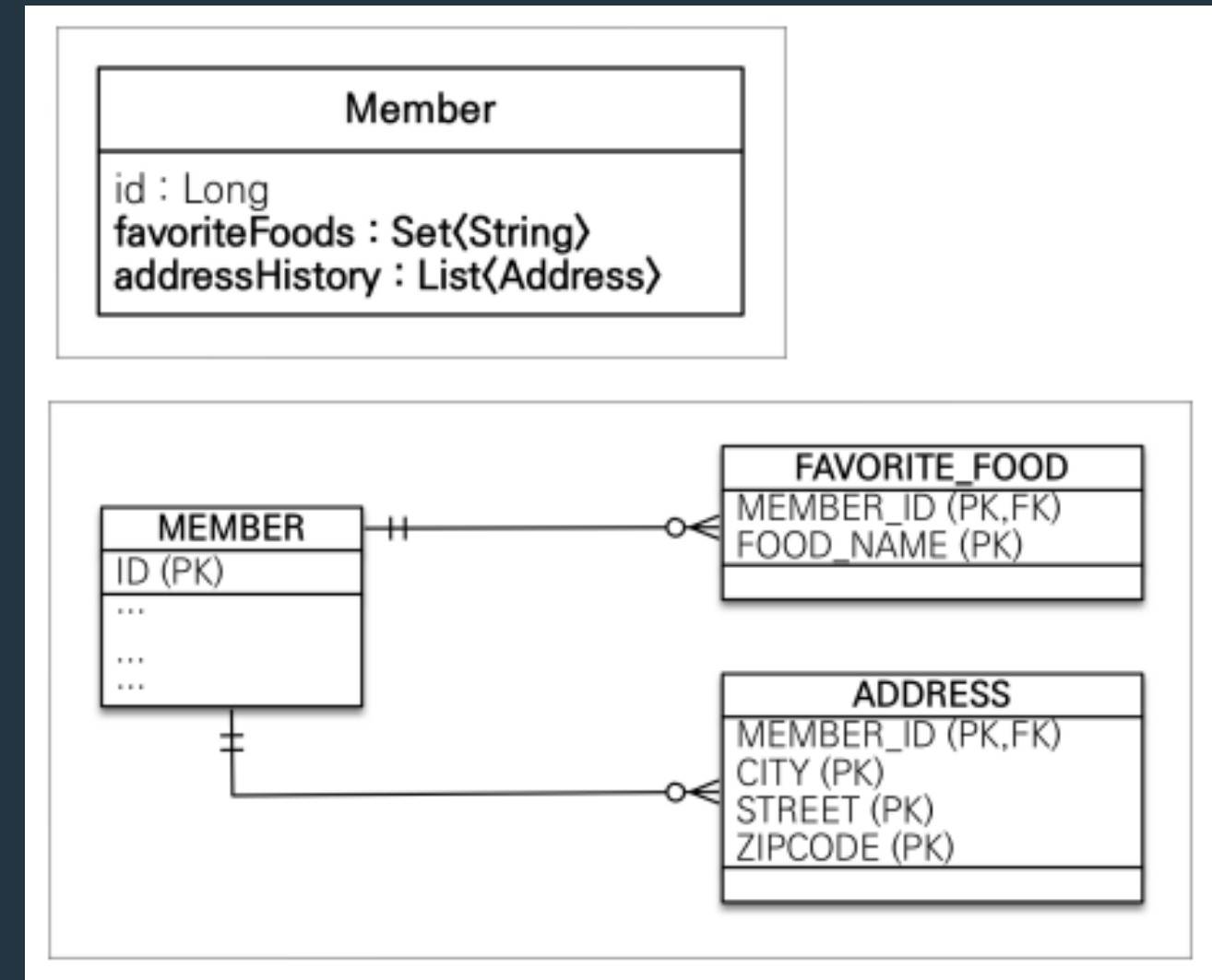
Address b = new Address("서울시")

a==b F

a.equals(b) T

9. 값 타입 - 값 타입 컬렉션

- 값 타입을 하나 이상 저장할 때 사용
- @ElementCollection, @CollectionTable 사용
- 데이터베이스는 컬렉션을 같은 테이블에 저장할 수 없다.
- 컬렉션을 저장하기 위한 별도의 테이블이 필요함



9. 값 타입 - 값 타입 컬렉션

<제약사항>

- 값 타입 컬렉션에 변경 사항이 발생하면, 주인 엔티티와 연관된 모든 데이터를 삭제하고, 값 타입 컬렉션에 있는 현재 값을 모두 다시 저장한다.
- 값 타입 컬렉션을 매핑하는 테이블은 모든 컬럼을 묶어서 기본키를 구성해야 함: null 입력X, 중복 저장X

<대안>

- 실무에서는 상황에 따라 값 타입 컬렉션 대신에 일대다 관계를 고려
- 일대다 관계를 위한 엔티티를 만들고, 여기에서 값 타입을 사용