

istence API

es a POJO persistence model for object-relational mapping. The Java
by the EJB 3.0 software expert group as part of JSR 220, but its use is
ponents. It can also be used directly by web applications and
side the Java EE platform, for example, in Java SE applications. See



Chapter 9 Value Type

Main point

JPA type division & How to use JPA Value type

9.1 기본값 타입 basic value type

- 식별자 값(@Id)이 없고 생명주기가 없다.
- 인스턴스 제거 시 값도 제거
- int, short, long, boolean, float, double 와 같이 값 타입
- Wrapper class
 - 객체 지만 자바에서 기본 타입처럼 사용 하므로 기본타입으로 정의 사용
 - Immutable (make not to happen side effect)
 - Immutable Object
 - 생성 시점 이후 값 변경 할 수 없음
 - Without Setter
 - Integer, String 불변 객체

```


@Entity
public class Member{
    @Id @GeneratedValue
    private Long id;

    private String name;
    private int age;

    //Memeber instance 제거시 같이 제거 됨
}

```

9.2 Embedded 타입(내장타입, 복합 값 타입)

 새로운 값 타입을 직접 정의

9.2 기본회원 엔티티

```

@Entity
public class Member{
    @Id @GeneratedValue
    private Long id;

    //근무기간
    @Temporal(TemporalType.DATE) java.util.Date startDate;
    @Temporal(TemporalType.DATE) java.util.Date endDate;

    private String city;
    private String street;
    private String zipcode;

}

```

9.3 값 타입 적용 회원 엔티티

```

@Entity
public class Member{
    @Id @GeneratedValue
    private Long id;
    private String name;

    // 값 타입을 사용하는 곳에 표시
    @Embedded Period workPeriod // 근무 기간
}

```

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

9.4 기간 임베디드 타입

```
@Embeddable // 값 타입을 정의하는 곳에 표시
public class Period {
    @Temporal(TemporalType.DATE) Date startDate;
    @Temporal(TemporalType.DATE) Date endDate;

    public boolean isWork(Date date){
        //..값 타입을 위한 메소드를 정의할 수 있다.
    }
}
```

9.5 주소 임베디드

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

9.2.1 내장 타입과 테이블 매핑

- 내장 타입을 사용하기 전과 후 매핑 테이블은 동일(테이블이 늘어나지 않음) 같다.
- 객체와 테이블을 아주 세밀하게 매핑 가능
- 잘 설계 한 ORM 테이블 수보다 클래스의 수가 더 많다.

9.2.2 내장 타입과 연관관계

1. 값 타입을 포함하거나 엔티티를 참조할 수 있다.
 - a. 엔티티는 공유될 수 있으므로 참조한다고 표현
 - b. 값 타입은 특정 주인에 소속되고 논리적인 개념상 공유 되지 않으므로 포함
2. @Embedded : 값 타입을 사용 하는 곳에 표시
3. @Embeddable(@Column 사용가능) : 값 타입을 정의 하는 곳에 표시

4. 둘 중 하나는 생략 가능(둘다 사용을 권장)
5. UML 컴포지션 관계?
6. 하이버네이트는 임베디드 타입을 컴포넌트라 한다.

9.2.3 @AttributeOverride : 속성 재정의

1. 매핑정보를 재정의

엔티티에 같은 임베디드 타입을 중복해서 사용하는 경우는 많지 않다.

엔티티에 설정해야 한다.

9.7 같은 임베디드 타입을 가지고 있는 회원

```
@Entity
public class Memeber {
    @Id @GeneratedValue
    private Long id;
    private String name;

    @Embedded Address homeAddr;
    @Embedded Address companyAddr;
}
```


```
@Entity
public class Memeber {
    @Id @GeneratedValue
    private Long id;
    private String name;

    @Embedded Address homeAddr;
    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name="city", column=@Column(name ="COMPANY_CITY")),
        @AttributeOverride(name="street", column=@Column(name ="COMPANY_STREET")),
        @AttributeOverride(name="zipcode", column=@Column(name ="COMPANY_ZIPCODE")),
    })
    Address companyAddr;
}
```

9.2.4 내장 타입과 Null

1. 내장 타입이 null 이면 매핑한 컬럼 값은 모두 null
-

9.3 값 타입과 불변 객체

 불변객체로 작성

9.3.1 값 타입 공유 참조

- 객체 세상을 조금이라도 단순화
- 단순하고 안전하게 다뤄야한다.
- 내장 타입 같은 값 타입을 여러 엔티티에서 공유하면 위험

9.3.2 값 타입 복사

- 인스턴스를 복사 ex) `address.clone()`
- 복사하지 않고 원본의 참조 값을 직접 넘기는 것을 막을 방법이 없음
- 기본타입이면 값을 복사, 객체면 참조를 넘긴다.
- 객체의 공유 참조는 피할 수 없다.
 - setter같은 수정자 메소드를 모두 제거하여 부작용을 방지 하자

9.3.3 불변객체

- 부작용? NO 값 타입
- 객체를 불변하게 생성하여 부작용 차단
- 불변객체도 객체 참조 값 공유를 피할 수 없음
- 참조값을 공유해도 값을 수정할 수 없으므로 부작용이 발생하지 않음
- 가장 간단한 방법 → 생성자로만 값을 설정(빌더도 가능)
- Integer, String은 자바가 제공하는 대표적인 불변 객체

9.4 값 타입의 비교

- 값 타입은 새로 정의된 equals()로 사용(전체 값을 비교)
- hashCode()도 재정의, 해시를 사용하는 컬렉션에서 문제가 발생할 수 있기때문

9.5 값 타입 컬렉션

- @ElementCollection, @CollectionTable : 값 타입을 하나 이상 저장할 때 사용
 - CollectionTable을 생략하면 기본값을 사용해서 매핑.
 1. 엔티티이름_컬렉션 속성이름 ex) addressHistory >> Member_addressHistory 테이블과 매핑

```
@Entity
public class Member{
    @Id @GeneratedValue
    private Long id;

    @Embedded Address homeAddress; //집 주소

    @ElementCollection
    @CollectionTable(name="FAVORITE_FOODS", joinColumns = @JoinColumn(name= "MEMBER_ID"))
    @Column(name="FOOD_NAME")
    private Set<String> favoriteFoods = new HashSet<String>();

    @ElementCollection
    @CollectionTable(name="ADDRESS", joinColumns = @JoinColumn(name= "MEMBER_ID"))
    private List<Address> addressHistory = new ArrayList<Address>();
}

@Embeddable
public class Address {
    @Column
    private String city;
    private String street;
    private String zipcode;
}
```

9.5.1 값 타입 컬렉션 사용

- 영속성 전이(cascade) + 고아 객체 제거(ORPHAN REMOVE) 기능을 필수로 가진다
고 볼 수 있다.
- 조회 시 패치 전략을 선택할 수 있다 기본은 LAZY

- 수정
 - 내장 값 타입 수정
 - 기본값 타입 컬렉션 수정 : 기존 값을 제거하고 새로운 값을 추가(자바의 String타입은 수정 불가)
 - 내장 값 타입 컬렉션 수정 : 기존 주소 삭제 새로운 주소를 등록

```

Memeber member = new Member();

//임베디드 값 타입
memeber.setHomeAddress(new Address("통영", "해수욕장", "660-123"));

//기본값 타입 컬렉션
member.getFavoriteFoods().add("닭볶음");
member.getFavoriteFoods().add("양장피");
member.getFavoriteFoods().add("튀바로우");

//임베디드 값 타입 컬렉션

member.getAddressHistory().add(new Address("서울", "송파", "123-123"));
member.getAddressHistory().add(new Address("서울", "서초", "012-123"));


em.persist(memeber);

```

9.5.2 값 타입 컬렉션의 제약사항

- 내장 값 타입 컬렉션
 - 관련된 모든 데이터 삭제 → 다시 전부 입력(최악)
 - 값 타입 컬렉션 대신 일대다 고려 (추가로 영속성 전이 + 고아 객체 제거 기능을 적용하면 값 타입 컬렉션처럼 사용가능)

9.6 정리

 불변 객체로 만들어야 안전

1. Entity 타입 특징

- 식별자가 있다.

- 생명주기가 있다.
 - 영속화/제거
 - 공유할 수 있다.

2. 값 타입 특징

- 식별자가 없다.
- 생명 주기를 Entity에 의존한다. (Entity 제거시 같이 제거)
- 공유하지 않는것이 안전
 - 사용해야 한다면 clone()
 - 오직 하나의 주인만이 관리?