10-5. 값 타입 컬렉션

값 타입을 컬렉션에 담아서 쓰는 것을 말함.
엔티티 컬렉션(`List<Child> list = new ArrayList<>();`) 은 해봤다

RDB는 기본적으로 컬렉션을 테이블에 저장할 수 있는게 없다.
=> 값 타입을 컬렉션하려면 따로 테이블을 뽑아야 한다.

```
@Embedded
private Address homeAddress;

@ElementCollection
@CollectionTable(name = "FAVORITE_FOOD", joinColumns =
@JoinColumn(name = "MEMBER_ID")) // @JoinColumn으로 외래키 잡기
@Column(name = "FOOD_NAME") // 임베디드 타입이 아니고 정의한게 아니라서...예외적
private Set<String> favoriteFoods = new HashSet<>();

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

값 타입을 하나 이상 저장하는 컬렉션을 만든다!
@ElementCollection, @CollectionTable 사용하여 테이블을 따로 만들기
따라서 외래키가 필요함!

값 타입의 조회

속해있는 엔티티에 종속된다!
속해있는 엔티티가 없어지면 같이 없어진다.
=> 영속성 전이 + 고아 객체 제거 기능을 디폴트로 가지고 있다.
=> 값 타입으로 만든 컬렉션도 지연 로딩 전략을 사용한다.

```
```

```
Member member = new Member();
member.setName("member1");
member.setHomeAddress(new Address("city1", "street", "zipcode"));

member.getFavoriteFoods().add("치킨");
member.getFavoriteFoods().add("족발");
member.getFavoriteFoods().add("피자");

member.getAddressHistory().add(new Address("old1", "street", "zipcode"));
member.getAddressHistory().add(new Address("old2", "street", "zipcode"));

em.persist(member);

em.flush();
em.clear();

Member findMember = em.find(Member.class, member.getId());

List<Address> addressHistory = findMember.getAddressHistory();
for (Address address : addressHistory) {
  System.out.println("address = " + address.getCity());
}

Set<String> favoriteFoods = findMember.getFavoriteFoods();
for (String favoriteFood : favoriteFoods) {
  System.out.println("foods = " + favoriteFood);
}

tx.commit();
```

findMember를 실행하면 값 타입들은 지연 로딩된다.

값 타입 수정

```
findMember.getFavoriteFoods().remove("치킨");
```

```
findMember.getFavoriteFoods().add("찌개");


findMember.getAddressHistory().remove(new Address("old1, "street", "10000");
findMember.getAddressHistory().add(new Address("old2, "street", "19999");
```

이렇게 통으로 갈아끼워줘야 한다. 왜냐면 값 타입도 객체이므로.

밑에 주소를 가지고 하는게 String이 아닌 임베디드 값 타입을 수정한 것인데, equals가 잘 구현되어있지 않으면 잘 안돌아간다.

➜ 이 때 Address 테이블이 완전히 지워지고 원래 있던 데이터 + 수정한 데이터 이렇게 두개가 insert된다.

=> 값 타입은 엔티티와 다르게 식별자 개념이 없어서 변경 사항이 발생하면 주인 엔티티와 연관된 모든 데이터를 삭제하고 원래 있던 데이터를 insert친다.
==> 그냥 쓰면 안됨! 완전히 다르게 풀어야된다! 결론은 Address를 아예 엔티티로 빼서 다대일 관계를 고려해야 한다.
===> 값 타입이 엔티티로 승급!

```
@OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
@JoinColumn(name = "MEMBER_ID")
private List<AddressEntity> addressHistory = new ArrayList<>();
```

값 타입 컬렉션을 쓰는 경우

값을 변경하지 않는 그냥 상수같은경우.

주소 이력 같은 경우는 엔티티..

정리

- **엔티티 타입의 특징**
  - 식별자O
  - 생명 주기 관리
  - 공유
- **값 타입의 특징**
  - 식별자X
  - 생명 주기를 엔티티에 의존
  - 공유하지 않는 것이 안전(복사해서 사용)
  - 불변 객체로 만드는 것이 안전

#JPA