학습 목표

- 1. 연관 관계 매핑의 종류와 엔티티 연관 관계 매핑을 설정하는 방법을 알아본다.
- 2. 매핑된 엔티티 조회 시 즉시 로딩과 지연 로딩의 차이점을 이해한다.

5.1 연관 관계 매핑 종류

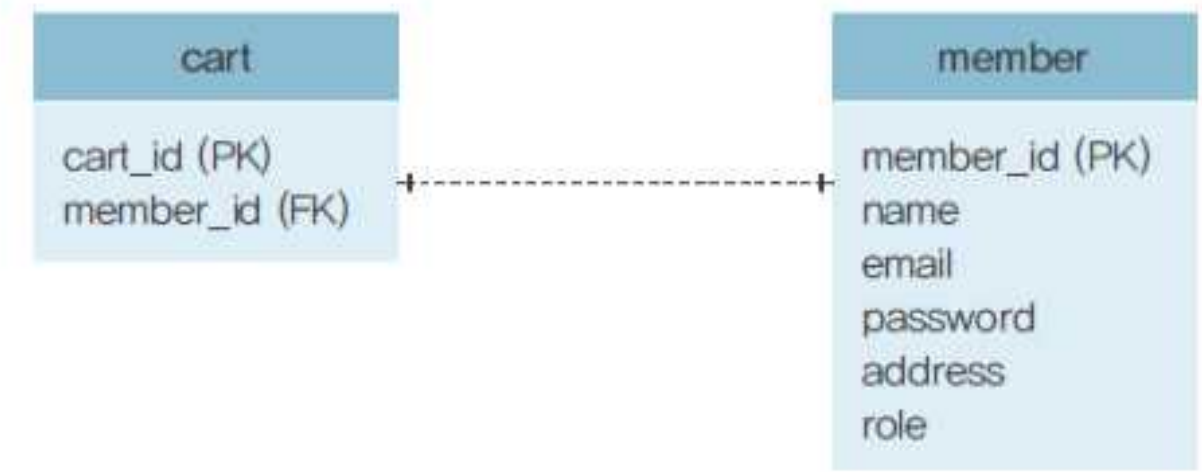
- 엔티티들은 대부분 다른 엔티티와 연관 관계를 맺으며, JPA에서는 엔티티에 연관 관계를 매핑해두고 필요할 때 해당 엔티티와 연관된 엔티티를 사용하여 좀 더 객체지향적으로 프로그래밍할 수 있도록 도와줌

- 연관 관계 매핑 종류
  - 일대일(1:1): @OneToOne
  - 일대다(1:N): @OneToMany
  - 다대일(N:1): @ManyToOne
  - 다대다(N:M): @ManyToMany

- 연관 관계 매핑 방향
  - 단방향
  - 양방향

5.1 연관 관계 매핑 종류 : 일대일 단방향 매핑

- 회원들은 각자 자신의 장바구니를 하나 갖고 있으며 장바구니 입장에서 보아도 자신과 매핑되는 한 명의 회원을 갖는 일대일 매핑 구조



[그림 5-1] 장바구니 - 회원 ERD(Entity Relationship Diagram)

5.1 연관 관계 매핑 종류 : 일대일 단방향 매핑

[함께 해보요 5-1] 장바구니 엔티티 설계하기

com.shop.entity.Cart.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05 import lombok.ToString;
06
07 import javax.persistence.*;
08
09 @Entity
10 @Table(name = "cart")
11 @Getter @Setter
12 @ToString
13 public class Cart {
14
15     @Id
16     @Column(name = "cart_id")
17     @GeneratedValue(strategy = GenerationType.AUTO)
18     private Long id;
19
20     @OneToOne ----------①
21     @JoinColumn(name="member_id") ----------②
22     private Member member;
23
24 }
```

* 장바구니 엔티티는 현재 회원 엔티티에 대한 정보를 알고 있음

* 회원 엔티티에는 장바구니(Cart) 엔티티와 관련된 소스가 전혀 없다는 것을 확인 가능.

즉, 장바구니 엔티티가 일방적으로 회원 엔티티를 참조하고 있는 **일대일 단방향 매핑**입니다.

5.1 연관 관계 매핑 종류 : 일대일 단방향 매핑

```
Hibernate:

    create table cart (
       cart_id bigint not null,
        member_id bigint,
        primary key (cart_id)
    )
```

[그림 5-2] cart 테이블 생성

```
Hibernate:

    alter table cart
       add constraint FKix170nytunweovf2v9137mx2o
       foreign key (member_id)
       references member
```

[그림 5-3] cart table foregin key 추가

* 애플리케이션을 실행시 콘솔창에 cart 테이블이 생성 및 외래키가 추가되는 쿼리문이 실행되는 것을 볼 수 있음

장바구니 엔티티 조회

[함께 해봐요 5-2] 장바구니 엔티티 조회 테스트하기(즉시 로딩)

com.shop.repository.CartRepository.java

```
01  package com.shop.repository;
02
03  import com.shop.entity.Cart;
04  import org.springframework.data.jpa.repository.JpaRepository;
05
06  public interface CartRepository extends JpaRepository<Cart, Long> {
07
08  }
```

장바구니 엔티티 조회

com.shop.entity.CartTest.java

```
01 package com.shop.entity;
02
03 import com.shop.dto.MemberFormDto;
04 import com.shop.repository.CartRepository;
05 import com.shop.repository.MemberRepository;
06 import org.junit.jupiter.api.DisplayName;
07 import org.junit.jupiter.api.Test;
08 import org.springframework.beans.factory.annotation.Autowired;
09 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.security.crypto.password.PasswordEncoder;
11 import org.springframework.test.context.TestPropertySource;
12 import org.springframework.transaction.annotation.Transactional;
13
14 import javax.persistence.EntityManager;
15 import javax.persistence.EntityNotFoundException;
16 import javax.persistence.PersistenceContext;
17
18 import static org.junit.jupiter.api.Assertions.assertEquals;
19
20 @SpringBootTest
21 @Transactional
22 @TestPropertySource(locations="classpath:application-test.properties")
```

장바구니 엔티티 조회

com.shop.entity.CartTest.java

```
01 package com.shop.entity;
02
03 import com.shop.dto.MemberFormDto;
04 import com.shop.repository.CartRepository;
05 import com.shop.repository.MemberRepository;
06 import org.junit.jupiter.api.DisplayName;
07 import org.junit.jupiter.api.Test;
08 import org.springframework.beans.factory.annotation.Autowired;
09 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.security.crypto.password.PasswordEncoder;
11 import org.springframework.test.context.TestPropertySource;
12 import org.springframework.transaction.annotation.Transactional;
13
14 import javax.persistence.EntityManager;
15 import javax.persistence.EntityNotFoundException;
16 import javax.persistence.PersistenceContext;
17
18 import static org.junit.jupiter.api.Assertions.assertEquals;
19
20 @SpringBootTest
21 @Transactional
22 @TestPropertySource(locations="classpath:application-test.properties")
23 class CartTest {
24
```

장바구니 엔티티 조회

```
25      @Autowired
26      CartRepository cartRepository;
27
28      @Autowired
29      MemberRepository memberRepository;
30
31      @Autowired
32      PasswordEncoder passwordEncoder;
33
34      @PersistenceContext
35      EntityManager em;
36
37      public Member createMember(){ ------------------------------------------ 1
38          MemberFormDto memberFormDto = new MemberFormDto();
39          memberFormDto.setEmail("test@email.com");
40          memberFormDto.setName("홍길동");
41          memberFormDto.setAddress("서울시 마포구 합정동");
42          memberFormDto.setPassword("1234");
43          return Member.createMember(memberFormDto, passwordEncoder);
44      }
45
```

장바구니 엔티티 조회

```
46      @Test
47      @DisplayName("장바구니 회원 엔티티 매핑 조회 테스트")
48      public void findCartAndMemberTest(){
49          Member member = createMember();
50          memberRepository.save(member);
51
52          Cart cart = new Cart();
53          cart.setMember(member);
54          cartRepository.save(cart);
55
56          em.flush();                                                    ❷
57          em.clear();                                                    ❸
58
59          Cart savedCart = cartRepository.findById(cart.getId())         ❹
60                  .orElseThrow(EntityNotFoundException::new);
61          assertEquals(savedCart.getMember().getId(), member.getId());  ❺
62      }
63
64  }
```

장바구니 엔티티 조회

```
46      @Test
47      @DisplayName("장바구니 회원 엔티티 매핑 조회 테스트")
48      public void findCartAndMemberTest(){
49          Member member = createMember();
50          memberRepository.save(member);
51
52          Cart cart = new Cart();
53          cart.setMember(member);
54          cartRepository.save(cart);
55
56          em.flush();                                                    ❷
57          em.clear();                                                    ❸
58
59          Cart savedCart = cartRepository.findById(cart.getId())         ❹
60                  .orElseThrow(EntityNotFoundException::new);
61          assertEquals(savedCart.getMember().getId(), member.getId());  ❺
62      }
63
64  }
```

장바구니 엔티티 조회

- cart테이블과 member 테이블을 조인해서 가져오는 쿼리문 실행
- cart 엔티티를 조회하면서 member 엔티티도 동시에 가져옴

```
Hibernate:
    select
        cart0_.cart_id as cart_id1_0_0_,
        cart0_.member_id as member_i2_0_0_,
        member1_.member_id as member_i1_3_1_,
        member1_.address as address2_3_1_,
        member1_.email as email3_3_1_,
        member1_.name as name4_3_1_,
        member1_.password as password5_3_1_,
        member1_.role as role6_3_1_
    from
        cart cart0_
    left outer join
        member member1_
            on cart0_.member_id=member1_.member_id
    where
        cart0_.cart_id=?
```

[그림 5-4] 장바구니 엔티티 조회 실행 쿼리

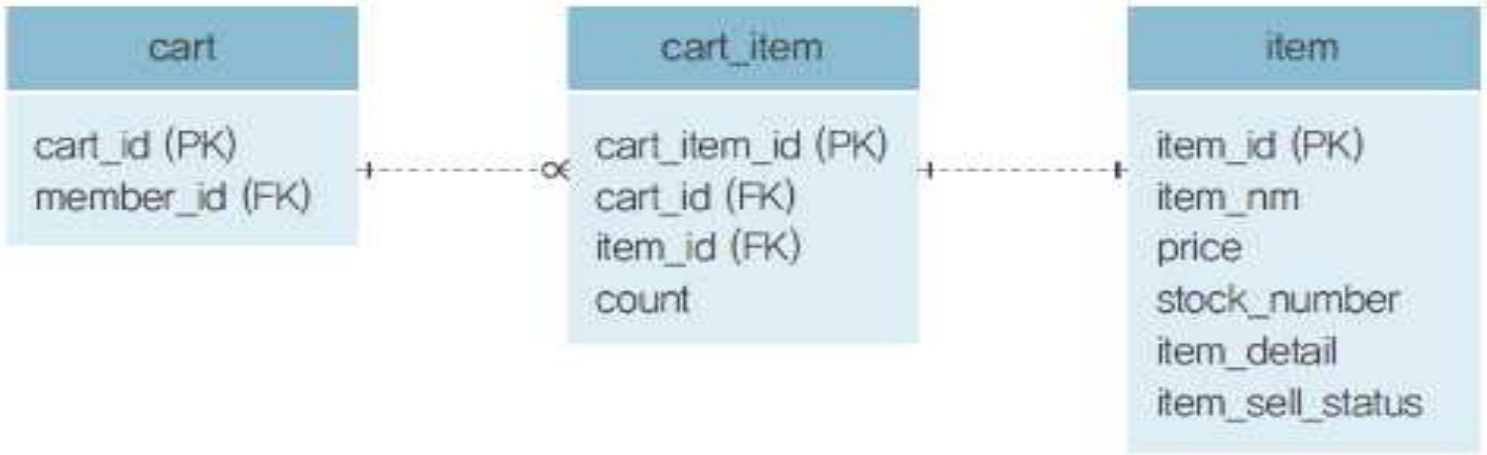
장바구니 엔티티 조회

- 즉시 로딩 : 엔티티를 조회할 때 해당 엔티티와 매핑된 엔티티도 한 번에 조회
- 일대일(@OneToOne), 다대일(@ManyToOne)로 매핑할 경우 즉시 로딩을 기본 Fetch 전략으로 설정

```
@OneToOne(fetch = FetchType.EAGER)
@JoinColumn(name="member_id")
private Member member;
```

5.1 연관 관계 매핑 종류 : 다대일 단방향 매핑

- 장바구니에는 고객이 관심이 있거나 나중에 사려는 상품들을 담아둠
- 하나의 장바구니에는 여러 개의 상품들이 들어갈 수 있음
- 하나의 상품은 여러 장바구니에 장바구니 상품으로 들어갈 수 있음



[그림 5-5] 장바구니 - 장바구니 상품 - 상품 ERD

5.1 연관 관계 매핑 종류 : 다대일 단방향 매핑

[함께 해봐요 5-3] 장바구니 아이템 엔티티 설계하기

com.shop.entity.CartItem.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05
06 import javax.persistence.*;
07
08 @Entity
09 @Getter @Setter
10 @Table(name="cart_item")
11 public class CartItem {
12
13     @Id
14     @GeneratedValue
15     @Column(name = "cart_item_id")
16     private Long id;
17
18     @ManyToOne
19     @JoinColumn(name="cart_id")
20     private Cart cart; ------------------------------------------ ❶
21
22     @ManyToOne
23     @JoinColumn(name = "item_id")
24     private Item item; ------------------------------------------ ❷
25
26     private int count; ------------------------------------------ ❸
27
28 }
```

5.1 연관 관계 매핑 종류 : 다대일 단방향 매핑

```
Hibernate:

    create table cart_item (
       cart_item_id bigint not null,
        count integer not null,
        cart_id bigint,
        item_id bigint,
        primary key (cart_item_id)
    ) engine=InnoDB
```

[그림 5-6] cart_item 테이블 생성

```
Hibernate:

    alter table cart_item
       add constraint FK1uobyhgl1wvgt1jpccia8xxs3
       foreign key (cart_id)
       references cart (cart_id)
Hibernate:

    alter table cart_item
       add constraint FKdljf497fwm1f8eb1h8t6n58u9
       foreign key (item_id)
       references item (item_id)
```

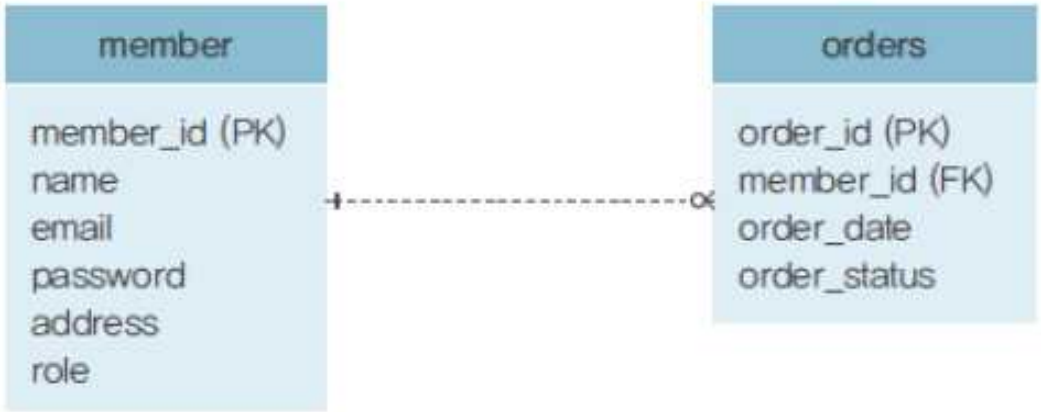
[그림 5-7] cart_item table foregin key 추가

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- 양방향 매핑이란 단방향 매핑이 2개 있는 구조
- 현재까지는 장바구니 상품 엔티티가(CartItem) 장바구니를(Cart) 참조하는 단방향 매핑.
- 장바구니 엔티티에 장바구니 상품 엔티티를 일대다 관계 매핑을 추가하여 양방향 매핑으로 변경가능

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- 주문과 주문 상품의 양방향 매핑 예제



[그림 5-8] 회원 – 주문 ERD

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- 주문 상태 Enum 생성

[함께 해봐요 5-4] 주문 도메인 엔티티 설계하기

com.shop.constatnt.OrderStatus.java

```
01 package com.shop.constant;
02
03 public enum OrderStatus {
04     ORDER, CANCEL
05 }
```

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

com.shop.entity.Order.java

```
01 package com.shop.entity;
02
03 import com.shop.constant.OrderStatus;
04 import lombok.Getter;
05 import lombok.Setter;
06
07 import javax.persistence.*;
08 import java.time.LocalDateTime;
09
10 @Entity
11 @Table(name = "orders") ------------------------------ ❶
12 @Getter @Setter
13 public class Order {
14
15     @Id @GeneratedValue
16     @Column(name = "order_id")
17     private Long id;
18
19     @ManyToOne
20     @JoinColumn(name = "member_id")
21     private Member member; ------------------------------ ❷
22
23     private LocalDateTime orderDate;  //주문일
24
25     @Enumerated(EnumType.STRING)
26     private OrderStatus orderStatus;  //주문상태
27
28     private LocalDateTime regTime;
29
30     private LocalDateTime updateTime;
31
32 }
```

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

com.shop.entity.OrderItem.java

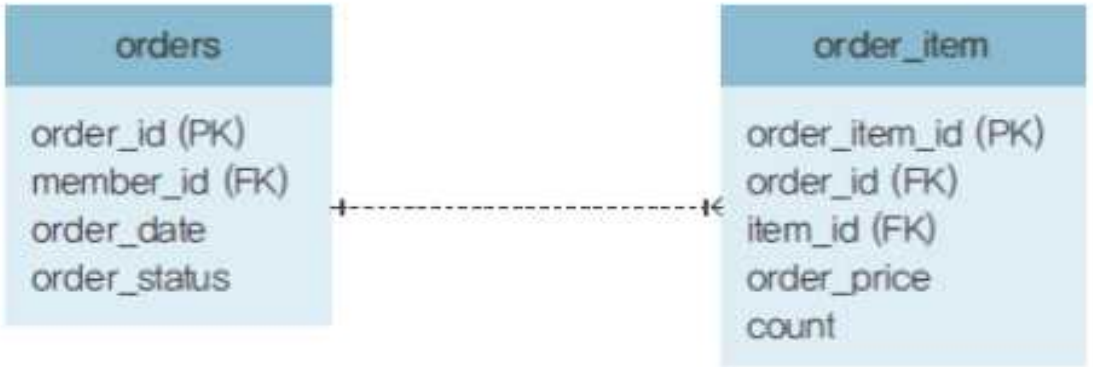
```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05
06 import javax.persistence.*;
07 import java.time.LocalDateTime;
08
09 @Entity
10 @Getter @Setter
11 public class OrderItem {
12
13     @Id @GeneratedValue
14     @Column(name = "order_item_id")
15     private Long id;
16
17     @ManyToOne
18     @JoinColumn(name = "item_id")
19     private Item item;
20
21     @ManyToOne
22     @JoinColumn(name = "order_id")
23     private Order order;
24
25     private int orderPrice;  //주문가격
26
27     private int count;  //수량
28
29     private LocalDateTime regTime;
30
31     private LocalDateTime updateTime;
32 }
```

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- 다대일과 일대다는 반대 관계
- 주문 상품 엔티티 기준에서 다대일 매핑이었으므로 주문 엔티티 기준에서는 주문 상품 엔티티와 일대다 관계로 매핑
- 양방향 매핑에서는 '**연관 관계 주인**'을 설정해야 한다는 점이 중요

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- ORDERS와 ORDER_ITEM 테이블을 ORDER_ID를 외래키로 조인하면 주문에 속한 상품이 어떤상품들이 있는지 알 수 있고, 주문 상품은 어떤 주문에 속하는지를 알 수 있음
- 즉, 테이블은 외래키 하나로 양방향 조회가 가능



[그림 5-9] 주문-주문 상품 ERD

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

- 엔티티는 테이블과 다릅니다. 엔티티를 양방향 연관 관계로 설정하면 객체의 참조는 둘인데 외래키는 하나이므로 둘 중 누가 외래키를 관리할지를 정해야 함.

- 연관 관계의 주인은 외래키가 있는 곳으로 설정
- 연관 관계의 주인이 외래키를 관리(등록, 수정, 삭제)
- 주인이 아닌 쪽은 연관 관계 매핑 시m appedBy 속성의 값으로 연관 관계의 주인을 설정
- 주인이 아닌 쪽은 읽기만 가능

5.1 연관 관계 매핑 종류 : 다대일/일대다 양방향 매핑

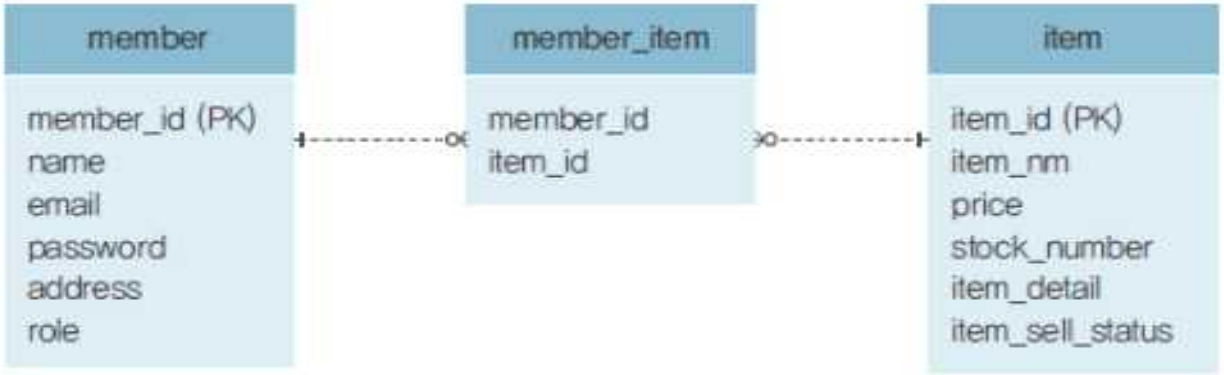
com.shop.entity.Order.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 import java.util.ArrayList;
06 import java.util.List;
07
08 @Entity
09 @Table(name = "orders")
10 @Getter @Setter
11 public class Order {
12
13     @Id @GeneratedValue
14     @Column(name = "order_id")
15     private Long id;
16
17     @ManyToOne
18     @JoinColumn(name = "member_id")
19     private Member member;
20
21     private LocalDateTime orderDate;  //주문일
22
23     @Enumerated(EnumType.STRING)
24     private OrderStatus orderStatus;  //주문상태
25
26     @OneToMany(mappedBy = "order")                    ①
27     private List<OrderItem> orderItems = new ArrayList<>();    ②
28
29     private LocalDateTime regTime;
30
31     private LocalDateTime updateTime;
32
33 }
```

- 외래키(order_id)가 order_item 테이블에 있으므로 연관 관계의 주인은 OrderItem 엔티티
- Order 엔티티가 주인이 아니므로 "mappedBy" 속성으로 연관 관계의 주인을 설정
- 속성의 값으로 "order"를 적어준 이유는 OrderItem에 있는 Order에 의해 관리된다는 의미로 해석.
- 즉, 연관 관계의 주인의 필드인 order를 mappedBy의 값으로 세팅

5.1 연관 관계 매핑 종류 : 다대다 매핑

- 실무에서는 사용하지 않는 매핑
- 관계형 데이터베이스는 정규화된 테이블 2개로 다대다를 표현할 수 없음
- 연결 테이블을 생성해서 다대다관계를 일대다, 다대일 관계로 풀어냄



[그림 5-10] 연결 테이블을 이용한 회원과 상품의 일대다, 다대일 관계

5.1 연관 관계 매핑 종류 : 다대다 매핑

- 객체는 테이블과 다르게 컬렉션을 사용해서 다대다 관계를 표현할 수 있음
- item을 리스트 형태로 가질 수 있으며, item 엔티티도 member를 리스트로 가질 수 있음

[그림 5-11] 회원, 상품 엔티티 다대다 관계 표현

5.1 연관 관계 매핑 종류 : 다대다 매핑

- @ManyToMany 어노테이션을 사용한다면 아래와 같이 사용 가능
- 다대다 매핑을 사용하지 않는 이유는 연결 테이블에는 컬럼을 추가할 수 없음
- 연결 테이블에는 조인 컬럼뿐 아니라 추가 컬럼들이 필요한 경우가 많음
- 엔티티를 조회할 때 member 엔티티에서 item을 조회하면 중간 테이블이 있기 때

```
01 public class Item {
02
03     @ManyToMany
04     @JoinTable(
05             name = "member_item",
06             joinColumns = @JoinColumn(name = "member_id"),
07             inverseJoinColumns = @JoinColumn(name = "item_id")
08     )
09     private List<Member> member;
10
11 }
```

5.2 영속성 전이

- 영속성 전이 : 엔티티의 상태를 변경할 때 해당 엔티티와 연관된 엔티티의 상태 변화를 전파시키는 옵션

- 부모는 One에 해당하고 자식은 Many에 해당

- (ex) Order 엔티티(부모)가 삭제되었을 때 해당 엔티티와 연관되어 있는 OrderItem 엔티티(자식)가 함께 삭제 되거나, Order 엔티티를 저장 할때 Order 엔티티에 담겨있던 OrderItem 엔티티를 한꺼번에 저장

5.2 영속성 전이

- 영속성 전이 옵션

CASCADE 종류	설명
PERSIST	부모 엔티티가 영속화될 때 자식 엔티티도 영속화
MERGE	부모 엔티티가 병합될 때 자식 엔티티도 병합
REMOVE	부모 엔티티가 삭제될 때 연관된 자식 엔티티도 삭제
REFRESH	부모 엔티티가 refresh되면 연관된 자식 엔티티도 refresh
DETACH	부모 엔티티가 detach 되면 연관된 자식 엔티티도 detach 상태로 변경
ALL	부모 엔티티의 영속성 상태 변화를 자식 엔티티에 모두 전이

5.2 영속성 전이

- 영속성 전이 옵션은 단일 엔티티에 완전히 종속적이고 부모 엔티티와 자식 엔

티티의 라이프 사이클이 유사할 때 cascade 옵션을 활용하기를 추천

- 무분별하게 사용할 경우 삭제되면 안될 데이터도 삭제될 수 있음

CASCADE 종류	설명
PERSIST	부모 엔티티가 영속화될 때 자식 엔티티도 영속화
MERGE	부모 엔티티가 병합될 때 자식 엔티티도 병합
REMOVE	부모 엔티티가 삭제될 때 연관된 자식 엔티티도 삭제
REFRESH	부모 엔티티가 refresh되면 연관된 자식 엔티티도 refresh
DETACH	부모 엔티티가 detach 되면 연관된 자식 엔티티도 detach 상태로 변경
ALL	부모 엔티티의 영속성 상태 변화를 자식 엔티티에 모두 전이

5.2 영속성 전이

[함께 해봐요 5-5] 주문 영속성 전이 테스트하기

com.shop.repository.OrderRepository.java

```
01 package com.shop.repository;
02
03 import com.shop.entity.Order;
04 import org.springframework.data.jpa.repository.JpaRepository;
05
06 public interface OrderRepository extends JpaRepository<Order, Long> {
07
08 }
```

5.2 영속성 전이

com.shop.entity.Order.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Table(name = "orders")
07 @Getter @Setter
08 public class Order {
09
10     .....코드 생략.....
11
12     @OneToMany(mappedBy = "order", cascade = CascadeType.ALL) ----------①
13     private List<OrderItem> orderItems = new ArrayList<>();
14
15 }
```

5.2 영속성 전이

com.shop.entity.OrderTest.java

```
01 package com.shop.entity;
02
03 import com.shop.constant.ItemSellStatus;
04 import com.shop.repository.ItemRepository;
05 import com.shop.repository.OrderRepository;
06 import org.junit.jupiter.api.DisplayName;
07 import org.junit.jupiter.api.Test;
08 import org.springframework.beans.factory.annotation.Autowired;
09 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.TestPropertySource;
11 import org.springframework.transaction.annotation.Transactional;
12
13 import javax.persistence.EntityManager;
14 import javax.persistence.EntityNotFoundException;
15 import javax.persistence.PersistenceContext;
16 import java.time.LocalDateTime;
17
18 import static org.junit.jupiter.api.Assertions.assertEquals;
19
20 @SpringBootTest
21 @TestPropertySource(locations="classpath:application-test.properties")
```

5.2 영속성 전이

```
22 @Transactional
23 class OrderTest {
24
25     @Autowired
26     OrderRepository orderRepository;
27
28     @Autowired
29     ItemRepository itemRepository;
30
31     @PersistenceContext
32     EntityManager em;
33
34     public Item createItem(){
35         Item item = new Item();
36         item.setItemNm("테스트 상품");
37         item.setPrice(10000);
38         item.setItemDetail("상세설명");
39         item.setItemSellStatus(ItemSellStatus.SELL);
40         item.setStockNumber(100);
41         item.setRegTime(LocalDateTime.now());
42         item.setUpdateTime(LocalDateTime.now());
43         return item;
44     }
```

5.2 영속성 전이

```
46      @Test
47      @DisplayName("영속성 전이 테스트")
48      public void cascadeTest(){
49
50          Order order = new Order();
51
52          for(int i=0;i<3;i++){
53              Item item = this.createItem();
54              itemRepository.save(item);
55              OrderItem orderItem = new OrderItem();
56              orderItem.setItem(item);
57              orderItem.setCount(10);
58              orderItem.setOrderPrice(1000);
59              orderItem.setOrder(order);
60              order.getOrderItems().add(orderItem);  ❶
61          }
62
63          orderRepository.saveAndFlush(order);  ❷
64          em.clear();  ❸
65
66          Order savedOrder = orderRepository.findById(order.getId())  ❹
67                  .orElseThrow(EntityNotFoundException::new);
68          assertEquals(3, savedOrder.getOrderItems().size());
69      }
70
71  }
```

5.2 영속성 전이

```
Hibernate:
    insert
    into
        orders
        (member_id, order_date, order_status, reg_time, update_time, order_id)
    values
        (?, ?, ?, ?, ?, ?)
```

[그림 5-13] flush 호출 시 orders 테이블 insert 쿼리문 실행

```
Hibernate:
    insert
    into
        order_item
        (reg_time, update_time, created_by, modified_by, count, item_id, order_id, order_price, order_item_id)
    values
        (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

[그림 5-14] flush 호출 시 order_item 테이블 insert 쿼리문 실행

* 영속성이 전이되면서 order에 담아 두었던 orderItem이 insert되는 것을 확인할 수 있음

* 총 3개의 orderItem을 담았두었으므로 3번의 insert 쿼리문이 실행

5.2 영속성 전이

- 테스트 코드 실행 결과 실제 조회되는 orderItem이 3개이므로 테스트가 정상적으로 통과

[그림 5-15] 영속성 전이 테스트 실행 결과

5.2 고아 객체 제거

- 고아 객체 : 부모 엔티티와 연관 관계가 끊어진 자식 엔티티
- 영속성 전이 기능과 같이 사용하면 부모 엔티티를 통해서 자식의 생명 주기를 관리할 수 있음

- 고아객체 제거 기능은 참조하는 곳이 하나일 때만 사용
- OrderItem 엔티티를 Order 엔티티가 아닌 다른 곳에서 사용하고 있다면 이 기능을 사용하면 안됨
- @OneToOne, @OneToMany 어노테이션에서 “orphanRemoval = true” 옵션 사용

5.2 고아 객체 제거

[함께 해봐요 5-6] 고아 객체 제거 테스트하기

com.shop.entity.Order.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Table(name = "orders")
07 @Getter @Setter
08 public class Order {
09
10     .....코드 생략.....
11
12     @OneToMany(mappedBy = "order", cascade = CascadeType.ALL ,
                  orphanRemoval = true)
13     private List<OrderItem> orderItems = new ArrayList<>();
14
15 }
```

5.2 고아 객체 제거

com.shop.entity.OrderTest.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 import com.shop.repository.MemberRepository;
06
07 @SpringBootTest
08 @TestPropertySource(locations="classpath:application-test.properties")
09 @Transactional
10 class OrderTest {
11
12     .....코드 생략.....
13
```

5.2 고아 객체 제거

```
14      @Autowired
15      MemberRepository memberRepository;
16
17      public Order createOrder(){ ------------------------------------------------ 1
18          Order order = new Order();
19
20          for(int i=0;i<3;i++){
21              Item item = createItem();
22              itemRepository.save(item);
23              OrderItem orderItem = new OrderItem();
24              orderItem.setItem(item);
25              orderItem.setCount(10);
26              orderItem.setOrderPrice(1000);
27              orderItem.setOrder(order);
28              order.getOrderItems().add(orderItem);
29          }
30
31          Member member = new Member();
32          memberRepository.save(member);
33
34          order.setMember(member);
35          orderRepository.save(order);
36          return order;
37      }
```

5.2 고아 객체 제거

```
39      @Test
40      @DisplayName("고아객체 제거 테스트")
41      public void orphanRemovalTest(){
42          Order order = this.createOrder();
43          order.getOrderItems().remove(0);  ②
44          em.flush();
45      }
46
47  }
```

5.2 고아 객체 제거

- flush()를 호출하면 콘솔창에 orderItem을 삭제하는 쿼리문이 출력되는 것을 확인 가능
- 부모 엔티티와 연관 관계가 끊어졌기 때문에 고아 객체를 삭제하는 쿼리문이 실행됨

```
Hibernate:
    delete
    from
        order_item
    where
        order_item_id=?
```

[그림 5-16] 고아 객체가 될 경우 delete 쿼리문 실행

5.3 즉시 로딩의 문제점

- 조인해서 가져오는 엔티티가 많아질수록 쿼리가 어떻게 실행될지 개발자가 예측하기 힘듬
- 사용하지 않는 엔티티도 한꺼번에 가져오기 때문에 성능상 문제가 발생할 수 있음
- 즉시 로딩 대신에 지연 로딩 방식 사용

5.3 즉시 로딩의 문제점

[함께 해봐요 5-7] 주문 엔티티 조회 테스트하기(즉시 로딩)

com.shop.repository.OrderItemRepository.java

```
01 package com.shop.repository;
02
03 import com.shop.entity.OrderItem;
04 import org.springframework.data.jpa.repository.JpaRepository;
05
06 public interface OrderItemRepository extends JpaRepository<OrderItem, Long> {
07
08 }
```

5.3 즉시 로딩의 문제점

com.shop.entity.OrderTest.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 import com.shop.repository.OrderItemRepository;
06
07 @SpringBootTest
08 @TestPropertySource(locations="classpath:application-test.properties")
09 @Transactional
10 class OrderTest {
11
12     .....코드 생략.....
13
14     @Autowired
15     OrderItemRepository orderItemRepository;
16
17     @Test
18     @DisplayName("지연 로딩 테스트")
19     public void lazyLoadingTest(){
20         Order order = this.createOrder();                          ❶
21         Long orderItemId = order.getOrderItems().get(0).getId();
22         em.flush();
23         em.clear();
24
25         OrderItem orderItem = orderItemRepository.findById(orderItemId)   ❷
26                 .orElseThrow(EntityNotFoundException::new);
27         System.out.println("Order class : " +
                               orderItem.getOrder().getClass());           ❸
28     }
29
30 }
```

5.3 즉시 로딩의 문제점

```
Hibernate:
    select
        orderitem0_.order_item_id as order_it1_4_0_,
        orderitem0_.count as count2_4_0_,
        orderitem0_.item_id as item_id6_4_0_,
        orderitem0_.order_id as order_id7_4_0_,
        orderitem0_.order_price as order_pr3_4_0_,
        orderitem0_.reg_time as reg_time4_4_0_,
        orderitem0_.update_time as update_t5_4_0_,
        item1_.item_id as item_id1_2_1_,
        item1_.item_detail as item_det2_2_1_,
        item1_.item_nm as item_nm3_2_1_,
        item1_.item_sell_status as item_sel4_2_1_,
        item1_.price as price5_2_1_,
        item1_.reg_time as reg_time6_2_1_,
        item1_.stock_number as stock_nu7_2_1_,
        item1_.update_time as update_t8_2_1_,
        order2_.order_id as order_id1_5_2_,
        order2_.member_id as member_i6_5_2_,
        order2_.order_date as order_da2_5_2_,
        order2_.order_status as order_st3_5_2_,
        order2_.reg_time as reg_time4_5_2_,
        order2_.update_time as update_t5_5_2_,
        member3_.member_id as member_i1_3_3_,
        member3_.address as address2_3_3_,
        member3_.email as email3_3_3_,
        member3_.name as name4_3_3_,
        member3_.password as password5_3_3_,
        member3_.role as role6_3_3_
    from
        order_item orderitem0_
    left outer join
        item item1_
            on orderitem0_.item_id=item1_.item_id
    left outer join
        orders order2_
            on orderitem0_.order_id=order2_.order_id
    left outer join
        member member3_
            on order2_.member_id=member3_.member_id
    where
        orderitem0_.order_item_id=?
```

* 즉시 로딩 사용시 연관된 모든 객체를 가져오기 때문에 성능상 문제가 발생할 수 있음

5.3 지연 로딩

com.shop.entity.OrderItem.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05
06 import javax.persistence.*;
07 import java.time.LocalDateTime;
08
09 @Entity
10 @Getter @Setter
11 public class OrderItem {
12
13     @Id @GeneratedValue
14     @Column(name = "order_item_id")
15     private Long id;
16
17     @ManyToOne(fetch = FetchType.LAZY)
18     @JoinColumn(name = "item_id")
19     private Item item;
20
21     @ManyToOne(fetch = FetchType.LAZY)
22     @JoinColumn(name = "order_id")
23     private Order order;
24
25     private int orderPrice;  //주문 가격
26
```

* 즉시 로딩 대신에 지연 로딩 방식으로 변경

5.3 지연 로딩

com.shop.entity.OrderItem.java

```java
package com.shop.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
@Getter @Setter
public class OrderItem {

    @Id @GeneratedValue
    @Column(name = "order_item_id")
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "item_id")
    private Item item;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "order_id")
    private Order order;

    private int orderPrice;  //주문 가격

    private int count;  //수량

    private LocalDateTime regTime;

    private LocalDateTime updateTime;
}
```

* 즉시 로딩 대신에 지연 로딩 방식으로 변경

5.3 지연 로딩

com.shop.entity.OrderTest.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @SpringBootTest
06 @TestPropertySource(locations="classpath:application-test.properties")
07 @Transactional
08 class OrderTest {
09
10     .....코드 생략.....
11
12     @Test
13     @DisplayName("지연 로딩 테스트")
14     public void lazyLoadingTest(){
15         Order order = this.createOrder();
16         Long orderItemId = order.getOrderItems().get(0).getId();
17         em.flush();
18         em.clear();
19
20         OrderItem orderItem = orderItemRepository.findById(orderItemId)
21                 .orElseThrow(EntityNotFoundException::new);
22         System.out.println("Order class : " + orderItem.getOrder().getClass()); ❶
23         System.out.println("==========================");
24         orderItem.getOrder().getOrderDate(); ❷
25         System.out.println("==========================");
26     }
27
28 }
```

5.3 지연 로딩

- 테스트 코드 실행 결과 orderItem 엔티티만 조회하는 쿼리문이 실행되는 것을 볼 수 있음

```
Hibernate:
    select
        orderitem0_.order_item_id as order_it1_4_0_,
        orderitem0_.reg_time as reg_time2_4_0_,
        orderitem0_.update_time as update_t3_4_0_,
        orderitem0_.created_by as created_4_4_0_,
        orderitem0_.modified_by as modified5_4_0_,
        orderitem0_.count as count6_4_0_,
        orderitem0_.item_id as item_id8_4_0_,
        orderitem0_.order_id as order_id9_4_0_,
        orderitem0_.order_price as order_pr7_4_0_
    from
        order_item orderitem0_
    where
        orderitem0_.order_item_id=?
```

[그림 5-17] 지연 로딩 설정 후 테스트 실행 시 조회 쿼리

5.3 지연 로딩

- 테스트 코드의 실행 결과 Order 클래스 조회 결과가 HibernateProxy라고 출력되는 것을 볼 수 있음

- 지연 로딩으로 설정하면 실제 엔티티 대신에 프록시 객체를 넣어둠

- 프록시 객체는 실제로 사용되기 전까지 데이터 로딩을 하지 않고, 실제 사용 시점에 조회 쿼리문이 실행

```
Order class : class com.shop.entity.Order$HibernateProxy$m4a15CKN
```

[그림 5-18] 지연 로딩 설정 후 OrderItem에 매핑된 Order 클래스 출력 결과

5.3 지연 로딩

- Order의 주문일(orderDate)을 조회할 때 select 쿼리문이 실행되는 것을 확인 가능

```
Hibernate:
    select
        order0_.order_id as order_id1_5_0_,
        order0_.member_id as member_i6_5_0_,
        order0_.order_date as order_da2_5_0_,
        order0_.order_status as order_st3_5_0_,
        order0_.reg_time as reg_time4_5_0_,
        order0_.update_time as update_t5_5_0_,
        member1_.member_id as member_i1_3_1_,
        member1_.address as address2_3_1_,
        member1_.email as email3_3_1_,
        member1_.name as name4_3_1_,
        member1_.password as password5_3_1_,
        member1_.role as role6_3_1_
    from
        orders order0_
    left outer join
        member member1_
            on order0_.member_id=member1_.member_id
    where
        order0_.order_id=?
```

[그림 5-19] order 주문일 조회 시 실행되는 조회 쿼리

5.3 지연 로딩

- 인텔리제이에서 @OneToMany 어노테이션을 <Ctrl>+마우스로 클릭하면 @OneToMany 어노테이션의 경우는 기본 FetchType이 LAZY 방식으로 되어 있는 것을 확인할 수 있음

```
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface OneToMany {
    Class targetEntity() default void.class;

    CascadeType[] cascade() default {};

    FetchType fetch() default FetchType.LAZY;

    String mappedBy() default "";

    boolean orphanRemoval() default false;
}
```

[그림 5-20] @OneToMany 어노테이션 default FetchType

5.3 지연 로딩

- 어떤 어노테이션은 즉시 로딩이고, 어떤 어노테이션은 지연 로딩인데 사람인 이상 헷갈릴 수 있기 때문에 연관 관계 매핑 어노테이션에 Fetch 전략을 LAZY로 직접 설정

[함께 해봐요 5-8] 엔티티 지연 로딩 설정하기

com.shop.entity.Cart.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Table(name = "cart")
07 @Getter @Setter
08 @ToString
09 public class Cart {
10
11     @Id
12     @Column(name = "cart_id")
13     @GeneratedValue(strategy = GenerationType.AUTO)
14     private Long id;
15
16     @OneToOne(fetch = FetchType.LAZY)
17     @JoinColumn(name="member_id")
18     private Member member;
19
20 }
```

5.3 지연 로딩

com.shop.entity.CartItem.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Getter @Setter
07 @Table(name="cart_item")
08 public class CartItem {
09
10     @Id
11     @GeneratedValue
12     @Column(name = "cart_item_id")
13     private Long id;
14
15     @ManyToOne(fetch = FetchType.LAZY)
16     @JoinColumn(name="cart_id")
17     private Cart cart;
18
19     @ManyToOne(fetch = FetchType.LAZY)
20     @JoinColumn(name = "item_id")
21     private Item item;
22
23     private int count;
24
25 }
```

5.3 지연 로딩

com.shop.entity.Order.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Table(name = "orders")
07 @Getter @Setter
08 public class Order {
09
10     @Id @GeneratedValue
11     @Column(name = "order_id")
12     private Long id;
13
14     @ManyToOne(fetch = FetchType.LAZY)
15     @JoinColumn(name = "member_id")
16     private Member member;
17
18     private LocalDateTime orderDate;  //주문일
19
20     @Enumerated(EnumType.STRING)
21     private OrderStatus orderStatus;  //주문상태
22
23     @OneToMany(mappedBy = "order", cascade = CascadeType.ALL
24             , orphanRemoval = true, fetch = FetchType.LAZY)
25     private List<OrderItem> orderItems = new ArrayList<>();
26
27     private LocalDateTime regTime;
28
29     private LocalDateTime updateTime;
30
31 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- 지금까지 설계한 Item, Order, OrderItem 엔티티에는 등록시간(regTime), 수정시간(updateTime) 멤버변수가 공통
- Spring Data Jpa에서는 Auditing 기능을 제공하여 엔티티가 저장 또는 수정될 때 자동으로 등록일, 수정일, 등록자, 수정자를 입력 가능
- Audit의 사전적 정의는 '감시하다' 즉, 엔티티의 생성과 수정을 감시

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- 현재 로그인한 사용자의 정보를 등록자와 수정자로 지정하기 위해서 AuditorAware 인터페이스를 구현한 클래스를 생성

[함께 해봐요 5-9] Auditing 기능을 활용한 데이터 추적하기

com.shop.config.AuditorAwareImpl.java

```
01 package com.shop.config;
02
03 import org.springframework.data.domain.AuditorAware;
04 import org.springframework.security.core.Authentication;
05 import org.springframework.security.core.context.SecurityContextHolder;
06
07 import java.util.Optional;
08
09 public class AuditorAwareImpl implements AuditorAware<String> {
10
11     @Override
12     public Optional<String> getCurrentAuditor() {
13         Authentication authentication =
   SecurityContextHolder.getContext().getAuthentication();
14         String userId = "";
15         if(authentication != null){
16             userId = authentication.getName();  ①
17         }
18         return Optional.of(userId);
19     }
20 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- AuditorAwareImpl 객체 Bean 등록

com.shop.config.AuditConfig.java

```
01 package com.shop.config;
02
03 import org.springframework.context.annotation.Bean;
04 import org.springframework.context.annotation.Configuration;
05 import org.springframework.data.domain.AuditorAware;
06 import org.springframework.data.jpa.repository.config.EnableJpaAuditing;
07
08 @Configuration
09 @EnableJpaAuditing ..... ❶
10 public class AuditConfig {
11
12     @Bean
13     public AuditorAware<String> auditorProvider() { ..... ❷
14         return new AuditorAwareImpl();
15     }
16 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

com.shop.entity.BaseTimeEntity.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05 import org.springframework.data.annotation.CreatedDate;
06 import org.springframework.data.annotation.LastModifiedDate;
07 import org.springframework.data.jpa.domain.support.AuditingEntityListener;
08
09 import javax.persistence.Column;
10 import javax.persistence.EntityListeners;
11 import javax.persistence.MappedSuperclass;
12 import java.time.LocalDateTime;
13
14 @EntityListeners(value = {AuditingEntityListener.class}) ------ ❶
15 @MappedSuperclass ------ ❷
16 @Getter @Setter
17 public abstract class BaseTimeEntity {
18
19     @CreatedDate ------ ❸
20     @Column(updatable = false)
21     private LocalDateTime regTime;
22
23     @LastModifiedDate ------ ❹
24     private LocalDateTime updateTime;
25
26 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

com.shop.entity.BaseEntity.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import org.springframework.data.annotation.CreatedBy;
05 import org.springframework.data.annotation.LastModifiedBy;
06 import org.springframework.data.jpa.domain.support.AuditingEntityListener;
07
08 import javax.persistence.Column;
09 import javax.persistence.EntityListeners;
10 import javax.persistence.MappedSuperclass;
11
12 @EntityListeners(value = {AuditingEntityListener.class})
13 @MappedSuperclass
14 @Getter
15 public abstract class BaseEntity extends BaseTimeEntity{
16
17     @CreatedBy
18     @Column(updatable = false)
19     private String createdBy;
20
21     @LastModifiedBy
22     private String modifiedBy;
23 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- Member 엔티티에 Auditing 기능 적용을 위하여 BaseEntity 상속

com.shop.entity.Member.java

```
01 package com.shop.entity;
02
03 .....기존 임포트 생략.....
04
05 @Entity
06 @Table(name="member")
07 @Getter @Setter
08 @ToString
09 public class Member extends BaseEntity{
10
11     .....코드 생략.....
12
13 }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

com.shop.entity.MemberTest.java

```
01 package com.shop.entity;
02
03 import com.shop.repository.MemberRepository;
04 import org.junit.jupiter.api.DisplayName;
05 import org.junit.jupiter.api.Test;
06 import org.springframework.beans.factory.annotation.Autowired;
07 import org.springframework.boot.test.context.SpringBootTest;
08 import org.springframework.security.test.context.support.WithMockUser;
09 import org.springframework.test.context.TestPropertySource;
10 import org.springframework.transaction.annotation.Transactional;
11
12 import javax.persistence.EntityManager;
13 import javax.persistence.EntityNotFoundException;
14 import javax.persistence.PersistenceContext;
15
16 @SpringBootTest
17 @Transactional
18 @TestPropertySource(locations="classpath:application-test.properties")
19 public class MemberTest {
20
21     @Autowired
22     MemberRepository memberRepository;
23
24     @PersistenceContext
25     EntityManager em;
26
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

```
27      @Test
28      @DisplayName("Auditing 테스트")
29      @WithMockUser(username = "gildong", roles = "USER")
30      public void auditingTest(){
31          Member newMember = new Member();
32          memberRepository.save(newMember);
33
34          em.flush();
35          em.clear();
36
37          Member member = memberRepository.findById(newMember.getId())
38                  .orElseThrow(EntityNotFoundException::new);
39
40          System.out.println("register time : " + member.getRegTime());
41          System.out.println("update time : " + member.getUpdateTime());
42          System.out.println("create member : " + member.getCreatedBy());
43          System.out.println("modify member : " + member.getModifiedBy());
44      }
45
46  }
```

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- member 엔티티를 저장할 때 등록자나 등록일을 지정해주지 않았지만 저장 시간과 현재 로그인된 계정의 이름으로 저장된 것을 확인 가능

```
register time : 2021-03-07T16:15:07.551722
update time : 2021-03-07T16:15:07.551722
create member : gildong
modify member : gildong
```

[그림 5-21] Auditing 테스트 실행 콘솔 출력 결과

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

- 나머지 엔티티도 BaseEntity를 상속받도록 수정
- 엔티티에 등록 시간(regTime)과 수정 시간(updateTime)이 멤버 변수로 있었다면 삭제 후 상속
- 반복적인 내용이므로 OrderItem 엔티티만 예시 코드로 작성
- Cart, CartItem, Item, Order도 똑같이 수정

5.4 Auditing을 이용한 엔티티 공통 속성 공통화

com.shop.entity.OrderItem.java

```
01 package com.shop.entity;
02
03 import lombok.Getter;
04 import lombok.Setter;
05
06 import javax.persistence.*;
07
08 @Entity
09 @Getter @Setter
10 public class OrderItem extends BaseEntity{ ①
11
12     @Id @GeneratedValue
13     @Column(name = "order_item_id")
14     private Long id;
15
16     @ManyToOne(fetch = FetchType.LAZY)
17     @JoinColumn(name = "item_id")
18     private Item item;
19
20     @ManyToOne(fetch = FetchType.LAZY)
21     @JoinColumn(name = "order_id")
22     private Order order;
23
24     private int orderPrice;  //주문 가격
25
26     private int count;  //수량
27
28     private LocalDateTime regTime;  //삭제
29
30     private LocalDateTime updateTime; //삭제
31
32 }
```

Thank you for your attention