도메인 모델과 테이블 설계

실전 예제를 통해 완성한 도메인 모델을 사용해서 웹 애플리케이션을 개발해보자. 복습하는 차원에서 전체 내용을 다시 한번 정리하겠다.

- 요구사항 분석

JPA로 실제 도메인 모델을 어떻게 구성하고 다양한 연관관계를 어떻게 테이블에 매핑하는지 이해하기 위해 작은 쇼핑몰을 설계하고 만들어보자. 먼저 요구사항을 분석하고 도메인 모델과 테이블을 설계하자.

요구사항은 다음과 같다.

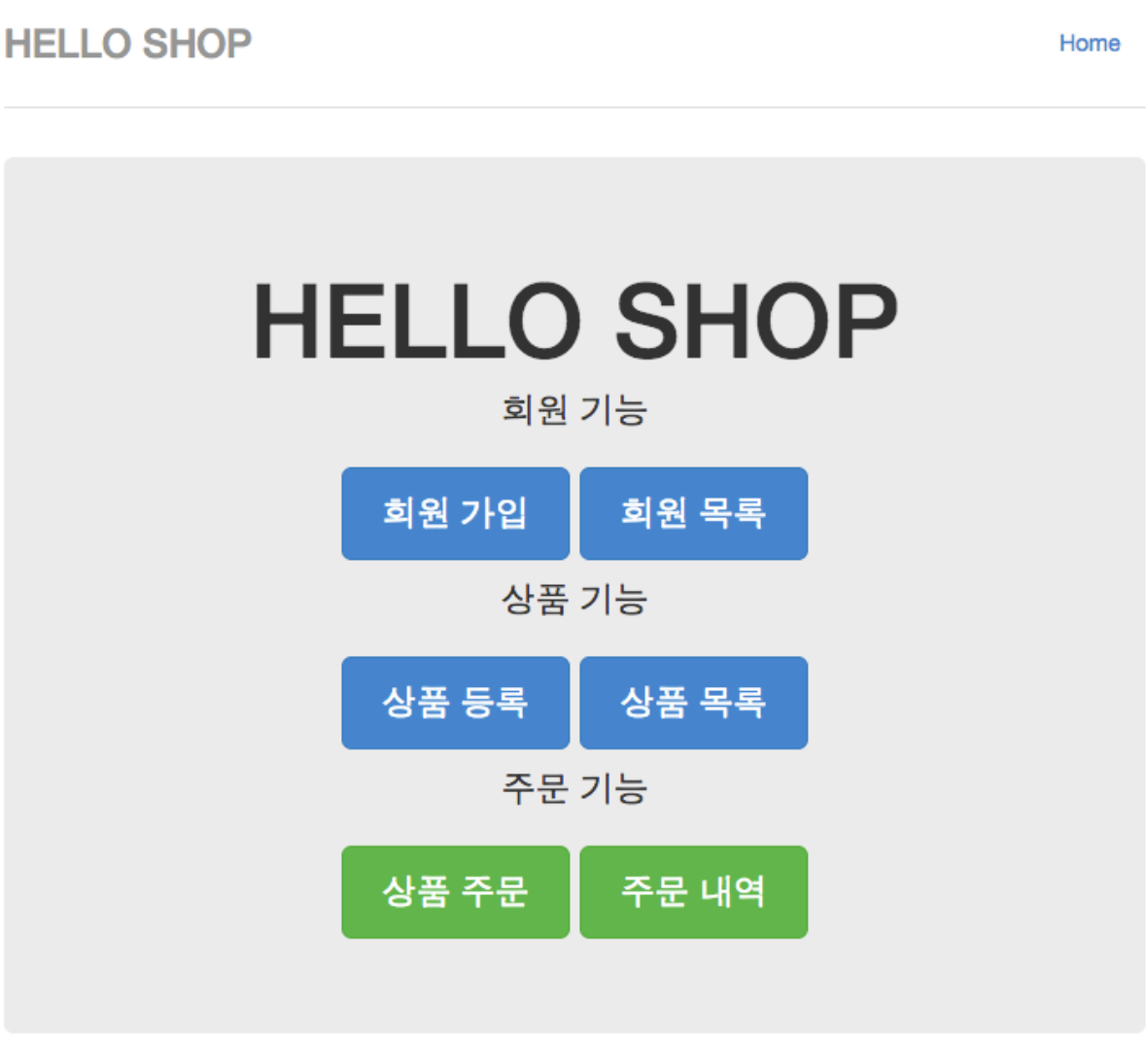

그림 - 메인 화면

- 회원 기능
  - 회원 등록
  - 회원 조회

- 상품 기능
  - 상품 등록
  - 상품 수정
  - 상품 조회

- 주문 기능
  - 상품 주문
  - 주문 내역 조회
  - 주문 취소

- 기타 요구사항

- 상품의 종류는 도서, 음반, 영화가 있다.
- 상품을 카테고리로 구분할 수 있다.
- 상품 주문시 배송 정보를 입력할 수 있다.

- 도메인 모델 설계

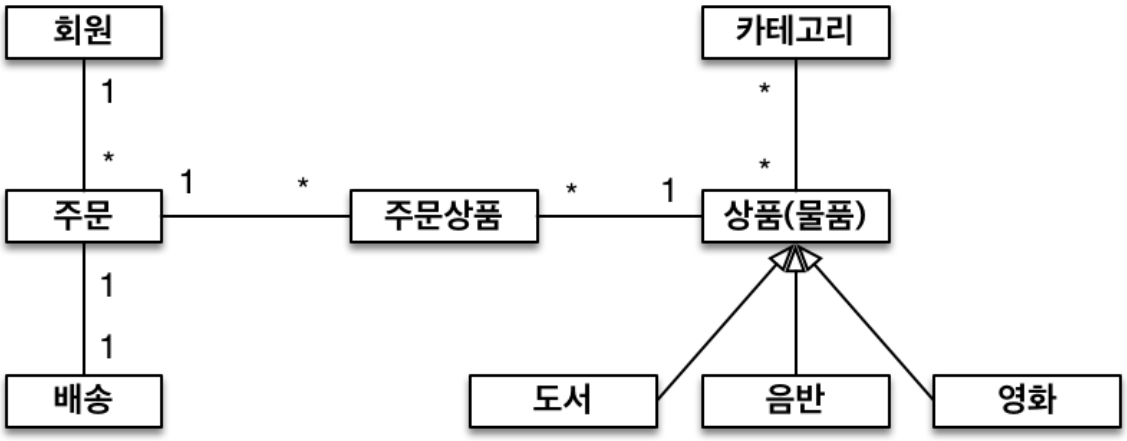

그림 - UML1 l

회원, 주문, 상품의 관계: 회원은 여러 상품을 주문할 수 있다. 한 번 주문할 때 여러 상품을 선택할 수 있으므로 주문과 상품은 다대다 관계다. 하지만 이런 다대다 관계는 관계형 데이터베이스는 물론이고 엔티티에서도 거의 사용하지 않는다. 따라서 [그림 - UML1]처럼 주문상품이라는 엔티티를 추가해서 다대다 관계를 일대다, 다대일 관계로 풀어냈다.

상품 분류: 상품은 도서, 음반, 영화로 구분되는데 상품이라는 공통 속성을 사용하므로 상속 구조로 표현했다.

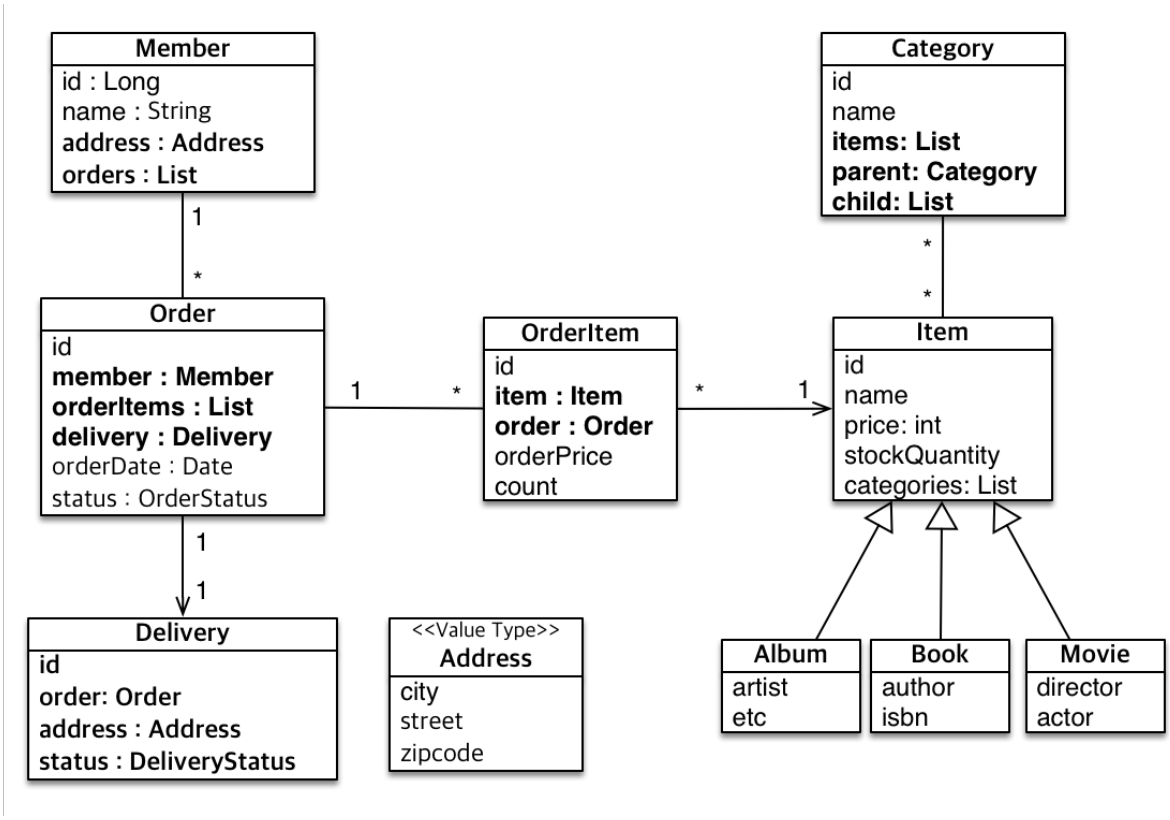

그림 - UML2 I 상세정보

이번에는 엔티티를 자세히 표현한 [그림 - UML2]를 분석해보자.

회원(`Member`): 이름과 주문한 상품리스트 그리고 임베디드 타입인 `Address` 를 가진다.

주문(`Order`): 한 번 주문시 여러 상품을 주문할 수 있으므로 주문과 주문상품(`OrderItem`)은 일대다 관계다. 주문은 상품을 주문한 회원과 배송 정보, 주문 날짜, 주문 상태(`status`)를 가지고 있다. 주문 상태는 열거형을 사용했는데 주문(`ORDER`), 취소(`CANCEL`)을 표현할 수 있다.

주문상품(`OrderItem`): 주문한 상품 정보와 주문 금액(`orderPrice`), 주문 수량(`count`) 정보를 가지고 있다.

상품(`Item`): 이름, 가격, 재고수량(`stockQuantity`)을 가지고 있다. 상품을 주문하면 재고수량이 줄어든다. 상품의 종류로는 앨범, 책, 영화가 있는데 각각은 사용하는 속성이 조금씩 다르다.

주소(`Address`): 값 타입(임베디드 타입)이다. 회원과 배송(`Delivery`)에서 사용한다.

배송(`Delivery`): 주문시 하나의 배송 정보를 생성한다. 주문과 배송은 일대일 관계다.

카테고리(`Category`): 상품과 다대다 관계를 맺는다.

- 테이블 설계

객체 도메인 모델을 매핑하기 위해 설계한 테이블은 다음과 같다.

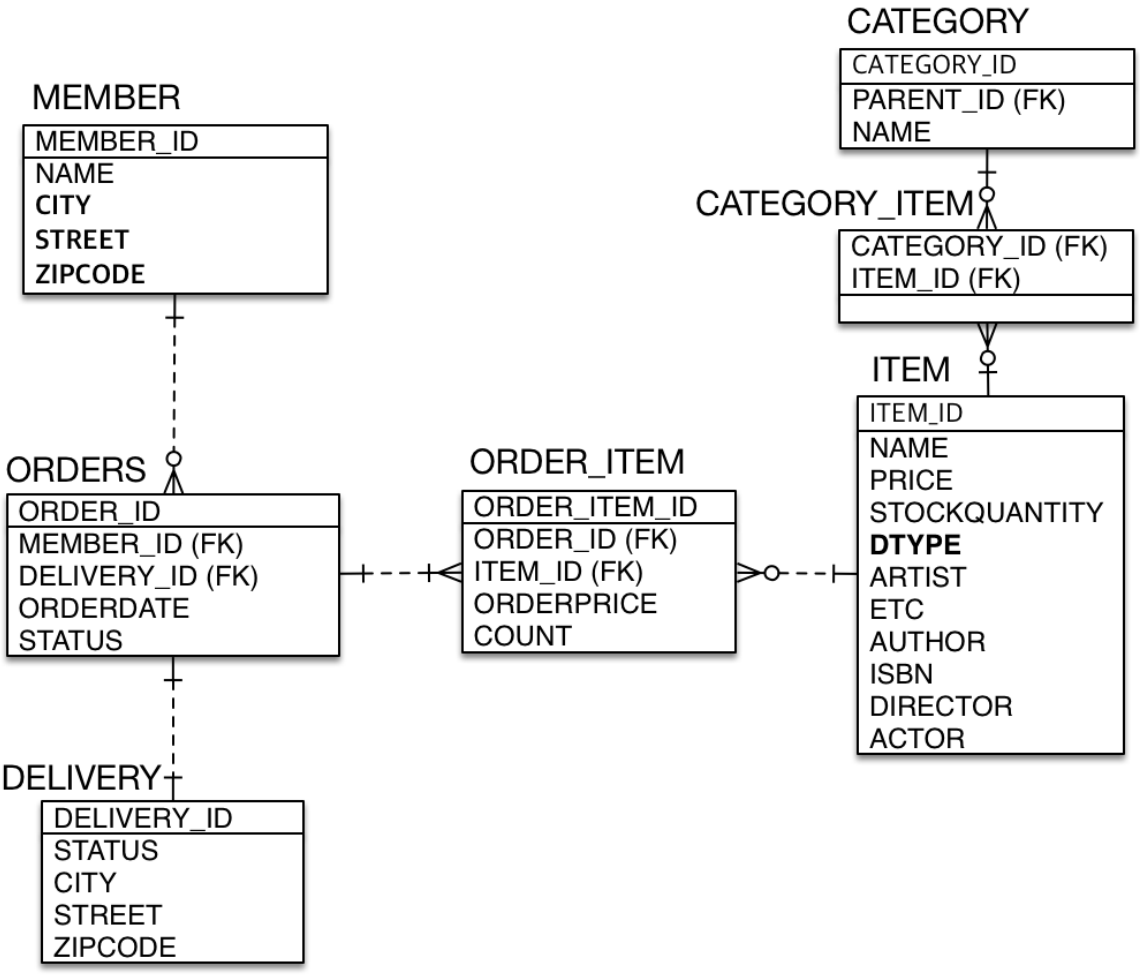

그림 - 테이블 ERD

[그림 - 테이블 ERD]를 보자.

MEMBER: 회원 엔티티의 `Address` 임베디드 타입 정보가 회원 테이블에 그대로 들어갔다. 이것은 `DELIVERY` 테이블도 마찬가지다.

ITEM: 앨범, 책, 영화 타입을 통합해서 하나의 테이블로 만들었다. `DTYPE` 컬럼으로 타입을 구분한다.

- 연관관계 정리

[그림 - UML2]와 [그림 - 테이블 ERD]를 보고 연관관계를 어떻게 매핑했는지 정리해보자.

회원과 주문: 일대다 양방향 관계다. 따라서 연관관계의 주인을 정해야 하는데 외래 키가 있는 주문이 연관관계의 주인이다. 그러므로 `Order.member` 를 `ORDERS.MEMBER_ID` 외래 키와 매핑한다.

주문상품과 주문: 다대일 양방향 관계다. 주문상품이 연관관계의 주인이다. 그러므로 `OrderItem.order`를 `ORDER_ITEM.ORDER_ID` 외래 키와 매핑한다.

주문상품과 상품: 다대일 단방향 관계다. `OrderItem.item`을 `ORDER_ITEM.ITEM_ID` 외래 키와 매핑한다.

주문과 배송: 일대일 단방향 관계다. `Order.delivery`를 `ORDERS.DELIVERY_ID` 외래 키와 매핑한다.

카테고리와 상품: `@ManyToMany`를 사용해서 매핑한다.

> **참고:** 실무에서는 `@ManyToMany`를 잘 사용하지 않지만 다양한 매핑 방법을 소개하려고 예제에 추가했다.

- 엔티티 클래스

실제 구현한 엔티티 클래스의 전체 코드는 다음과 같다. 이론 편을 충분히 이해했으면 분석하는 데 큰 어려움은 없을 것이다. 다양한 엔티티 매핑 사례를 담고 있으므로 UML, ERD 그림과 비교하면서 천천히 분석해보길 바란다.

====== 회원 엔티티 ======

```
package jpabook.jpashop.domain;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Member {

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

    private String name;

    @Embedded
    private Address address;

    @OneToMany(mappedBy = "member")
    private List<Order> orders = new ArrayList<Order>();
```

```
    //Getter, Setter
    ...
}
```

====== 주문 엔티티 ======

```
package jpabook.jpashop.domain;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

@Entity
@Table(name = "ORDERS")
public class Order {

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "MEMBER_ID")
    private Member member;      //주문 회원

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
    private List<OrderItem> orderItems = new ArrayList<OrderItem>();

    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "DELIVERY_ID")
    private Delivery delivery;  //배송정보

    private Date orderDate;     //주문시간

    @Enumerated(EnumType.STRING)
    private OrderStatus status;//주문상태

    //==연관관계 메서드==//
    public void setMember(Member member) {
        this.member = member;
        member.getOrders().add(this);
    }

    public void addOrderItem(OrderItem orderItem) {
        orderItems.add(orderItem);
        orderItem.setOrder(this);
    }
```

```
    public void setDelivery(Delivery delivery) {
        this.delivery = delivery;
        delivery.setOrder(this);
    }

    //Getter, Setter
    ...
}
```

====== 주문상품 엔티티 ======

```
package jpabook.jpashop.domain;

import jpabook.jpashop.domain.item.Item;
import javax.persistence.*;

@Entity
@Table(name = "ORDER_ITEM")
public class OrderItem {

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ITEM_ID")
    private Item item;      //주문 상품

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

    private int orderPrice; //주문 가격
    private int count;      //주문 수량

    //Getter, Setter
    ...
}
```

====== 상품 엔티티 ======

```
package jpabook.jpashop.domain.item;

import jpabook.jpashop.domain.Category;
import jpabook.jpashop.exception.NotEnoughStockException;
import javax.persistence.*;
```

```
import java.util.ArrayList;
import java.util.List;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "DTYPE")
public abstract class Item {

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

    private String name;
    private int price;
    private int stockQuantity;

    @ManyToMany(mappedBy = "items")
    private List<Category> categories = new ArrayList<Category>();

    //Getter, Setter
    ...
}
```

====== 상품 - 도서 엔티티 ======

```
package jpabook.jpashop.domain.item;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("B")
public class Book extends Item {

    private String author;
    private String isbn;

    //Getter, Setter
    ...
}
```

====== 상품 - 음반 엔티티 ======

```
package jpabook.jpashop.domain.item;

import javax.persistence.DiscriminatorValue;
```

```
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("A")
public class Album extends Item {

    private String artist;
    private String etc;

    //Getter, Setter
    ...
}
```

====== 상품 - 영화 엔티티 ======

```
package jpabook.jpashop.domain.item;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("M")
public class Movie extends Item {

    private String director;
    private String actor;

    //Getter, Setter
    ...
}
```

====== 배송 엔티티 ======

```
package jpabook.jpashop.domain;

import javax.persistence.*;

@Entity
public class Delivery {

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

    @OneToOne(mappedBy = "delivery")
    private Order order;
```

```
    @Embedded
    private Address address;

    @Enumerated(EnumType.STRING)
    private DeliveryStatus status; //ENUM [READY(준비), COMP(배송)]

    //Getter, Setter
    ...
}
```

====== 카테고리 엔티티 ======

```
package jpabook.jpashop.domain;

import jpabook.jpashop.domain.item.Item;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Category {

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

    private String name;

    @ManyToMany
    @JoinTable(name = "CATEGORY_ITEM",
            joinColumns = @JoinColumn(name = "CATEGORY_ID"),
            inverseJoinColumns = @JoinColumn(name = "ITEM_ID"))
    private List<Item> items = new ArrayList<Item>();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "PARENT_ID")
    private Category parent;

    @OneToMany(mappedBy = "parent")
    private List<Category> child = new ArrayList<Category>();

    //==연관관계 메서드==//
    public void addChildCategory(Category child) {
        this.child.add(child);
        child.setParent(this);
    }
```

```
    //Getter, Setter
    ...
}
```

====== 주소 값 타입 ======

```
package jpabook.jpashop.domain;

import javax.persistence.Embeddable;

@Embeddable
public class Address {

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

    //Getter, Setter
    ...
}
```

엔티티와 테이블 설계를 완성했다. 이 엔티티를 기반으로 실제 애플리케이션을 개발해보자.

> **참고: 기본 키 이름과 엔티티 식별자 이름**
>
> 테이블의 기본 키 컬럼 이름은 해당 테이블 이름을 포함하도록 했다. 예를 들어 `MEMBER` 테이블의 기본 키는 `MEMBER_ID` 로 지었다. 반면에 엔티티의 식별자 명은 모두 `id` 로 지었다. 왜냐하면 엔티티는 연관된 엔티티를 참조할 때 타입이 있으므로 엔티티의 식별자를 단순히 `id` 로 지어도 어떤 엔티티를 참조하는지 쉽게 알 수 있지만, 테이블의 기본 키나 외래 키는 단순히 값만 저장하지 어떤 테이블과 관계가 있는지 외래 키 제약조건을 참고하지 않으면 알 수 없다. 그리고 외래 키 이름은 관례로 기본 키 이름을 그대로 사용하는 경우가 많다. 따라서 테이블의 기본 키 이름은 편의상 테이블 이름을 포함하는 것이 유지 보수하기 편리하다.
>
> 따라서 다음과 같이 매핑했다.
>
> ```
> @Id @GeneratedValue
> @Column(name = "MEMBER_ID")
> private Long id;
> ```
