```
#      ddl-auto: create
#    properties:
#      hibernate:
         #        show_sql: true
#         format_sql: true
#    open-in-view: false

logging.level:
  org.hibernate.SQL: debug
#  org.hibernate.type: trace
```

이제 테스트에서 스프링을 실행하면 이 위치에 있는 설정 파일을 읽는다.
(만약 이 위치에 없으면 `src/resources/application.yml` 을 읽는다.)

스프링 부트는 datasource 설정이 없으면, 기본적을 메모리 DB를 사용하고, driver-class도 현재 등록된 라이브러리를 보고 찾아준다. 추가로 `ddl-auto` 도 `create-drop` 모드로 동작한다. 따라서 데이터소스나, JPA 관련된 별도의 추가 설정을 하지 않아도 된다.

상품 도메인 개발

구현 기능

- 상품 등록
- 상품 목록 조회
- 상품 수정

순서

- 상품 엔티티 개발(비즈니스 로직 추가)
- 상품 리포지토리 개발
- 상품 서비스 개발
- 상품 기능 테스트

상품 엔티티 개발(비즈니스 로직 추가)

상품 엔티티 코드

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

```java
import jpabook.jpashop.exception.NotEnoughStockException;
import lombok.Getter;
import lombok.Setter;

import jpabook.jpashop.domain.Category;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "dtype")
@Getter @Setter
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>();

    //==비즈니스 로직==//
    public void addStock(int quantity) {
        this.stockQuantity += quantity;
    }

    public void removeStock(int quantity) {
        int restStock = this.stockQuantity - quantity;
        if (restStock < 0) {
            throw new NotEnoughStockException("need more stock");
        }
        this.stockQuantity = restStock;
    }
```

변경해야하는 stockQuantity을 가지고 있는 엔티티(클래스). 전체적인 응집도를 위해서 엔티티안에서 비즈니스로직을 구현하는것도 좋은 방법이다.

```
}
```

예외 추가

```
package jpabook.jpashop.exception;

public class NotEnoughStockException extends RuntimeException {

    public NotEnoughStockException() {
    }

    public NotEnoughStockException(String message) {
        super(message);
    }

    public NotEnoughStockException(String message, Throwable cause) {
        super(message, cause);
    }

    public NotEnoughStockException(Throwable cause) {
        super(cause);
    }

}
```

비즈니스 로직 분석

- `addStock()` 메서드는 파라미터로 넘어온 수만큼 재고를 늘린다. 이 메서드는 재고가 증가하거나 상품 주문을 취소해서 재고를 다시 늘려야 할 때 사용한다.
- `removeStock()` 메서드는 파라미터로 넘어온 수만큼 재고를 줄인다. 만약 재고가 부족하면 예외가 발생한다. 주로 상품을 주문할 때 사용한다.

상품 리포지토리 개발

상품 리포지토리 코드

```java
package jpabook.jpashop.repository;

import jpabook.jpashop.domain.item.Item;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Repository;

import javax.persistence.EntityManager;
import java.util.List;

@Repository
@RequiredArgsConstructor
public class ItemRepository {

    private final EntityManager em;

    public void save(Item item) {  jpa에 저장하기 전까지 id값이 없음.
        if (item.getId() == null) { 새로 생성한 객체라면?
            em.persist(item);
        } else {
            em.merge(item);  업뎃개념임
        }
    }

    public Item findOne(Long id) {
        return em.find(Item.class, id);
    }

    public List<Item> findAll() {
        return em.createQuery("select i from Item
i",Item.class).getResultList();      여러개 찾는건 쿼리사용!
    }
}
```

기능 설명

- `save()`
  - `id`가 없으면 신규로 보고 `persist()` 실행
  - `id`가 있으면 이미 데이터베이스에 저장된 엔티티를 수정한다고 보고, `merge()`를 실행, 자세한 내용

은 뒤에 웹에서 설명(그냥 지금은 저장한다 정도로 생각하자)

상품 서비스 개발

상품 서비스 코드

```
package jpabook.jpashop.service;

import jpabook.jpashop.domain.item.Item;
import jpabook.jpashop.repository.ItemRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

@Service
@Transactional(readOnly = true)
@RequiredArgsConstructor
public class ItemService {

    private final ItemRepository itemRepository;

    @Transactional
    public void saveItem(Item item) {
        itemRepository.save(item);
    }

    public List<Item> findItems() {
        return itemRepository.findAll();
    }

    public Item findOne(Long itemId) {
        return itemRepository.findOne(itemId);
    }
}
```

상품 서비스는 상품 리포지토리에 단순히 위임만 하는 클래스