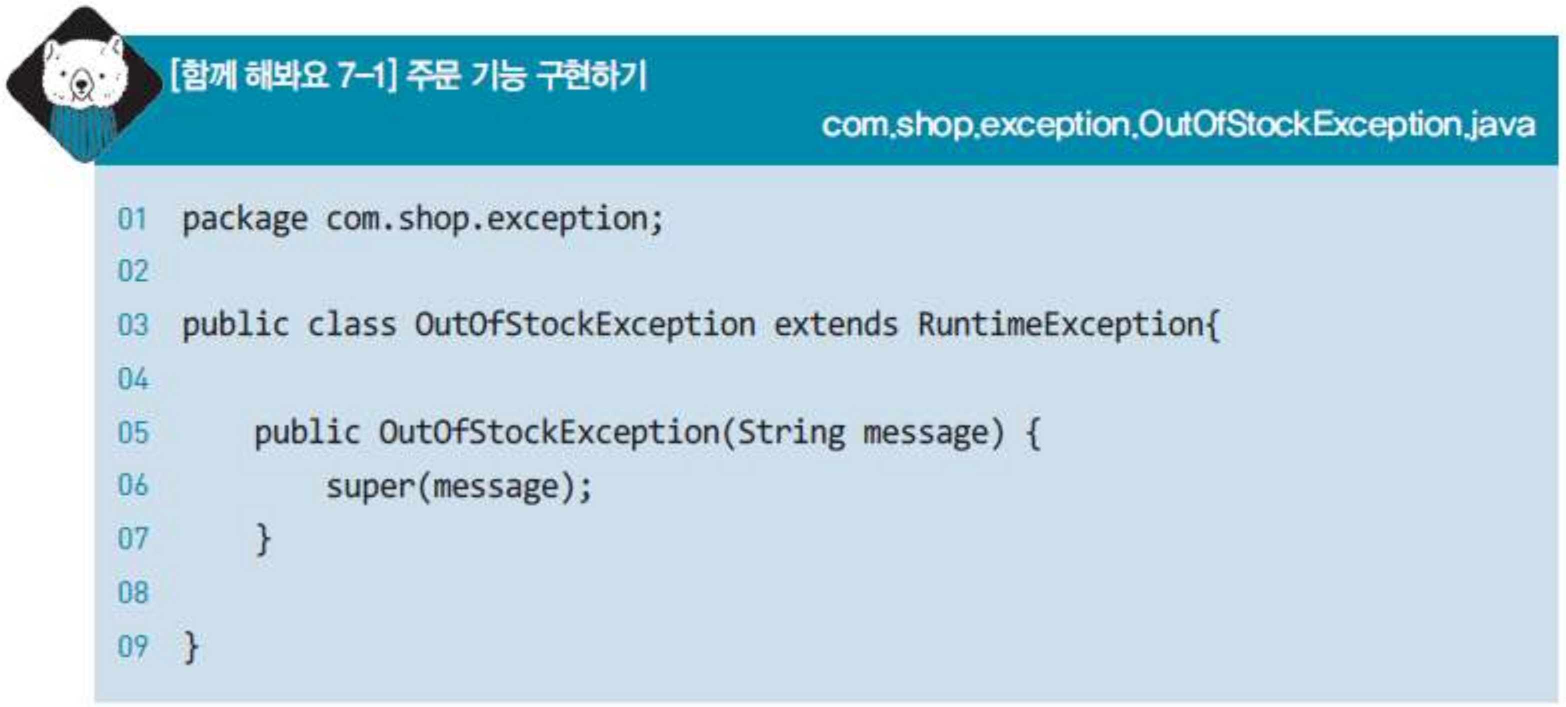
학습 목표

- 1. 주문, 주문이력 조회, 주문 취소 기능 구현을 통해서 주문 프로세스를 학습한다.
- 2. Spring Data JPA를 이용하여 주문 데이터 조회 시 조회를 최적화하는 방법을 학습한다.

7.1 주문 기능 구현

1. 주문하면 현재 상품의 재고에서 주문 수량만큼 재고 감소
2. 상품의 주문 수량보다 재고의 수가 적을 때 발생시킬 exception을 정의



[함께 해봐요 7-1] 주문 기능 구현하기

com.shop.exception.OutOfStockException.java

```java
package com.shop.exception;

public class OutOfStockException extends RuntimeException{

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

}
```

7.1 주문 기능 구현

com.shop.entity.Item.java

```java
package com.shop.entity;

.....기존 임포트 생략.....

import com.shop.exception.OutOfStockException;

@Entity
@Table(name="item")
@Getter @Setter
@ToString
public class Item extends BaseEntity{

    .....코드 생략.....

    public void removeStock(int stockNumber){
        int restStock = this.stockNumber - stockNumber; ❶
        if(restStock<0){
            throw new OutOfStockException("상품의 재고가 부족 합니다. (현재 재고 수량: " + this.stockNumber + ")"); ❷
        }
        this.stockNumber = restStock; ❸
    }

}
```

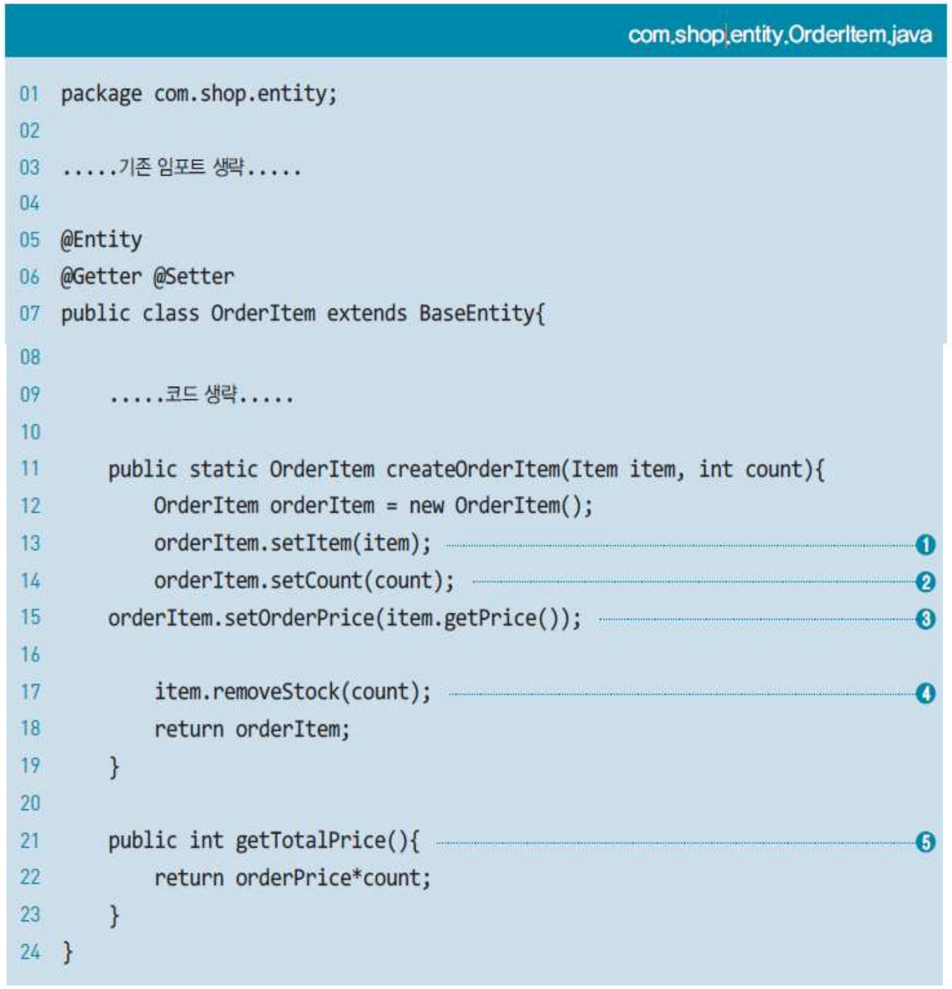
7.1 주문 기능 구현

com.shop.entity.Item.java

```java
package com.shop.entity;

.....기존 임포트 생략.....

import com.shop.exception.OutOfStockException;

@Entity
@Table(name="item")
@Getter @Setter
@ToString
public class Item extends BaseEntity{

    .....코드 생략.....

    public void removeStock(int stockNumber){
        int restStock = this.stockNumber - stockNumber; ❶
        if(restStock<0){
            throw new OutOfStockException("상품의 재고가 부족 합니다.
(현재 재고 수량: " + this.stockNumber + ")"); ❷
        }
        this.stockNumber = restStock; ❸
    }

}
```

7.1 주문 기능 구현

- 주문할 상품과 주문 수량을 통해 OrderItem 객체를 만드는 메소드를 작성

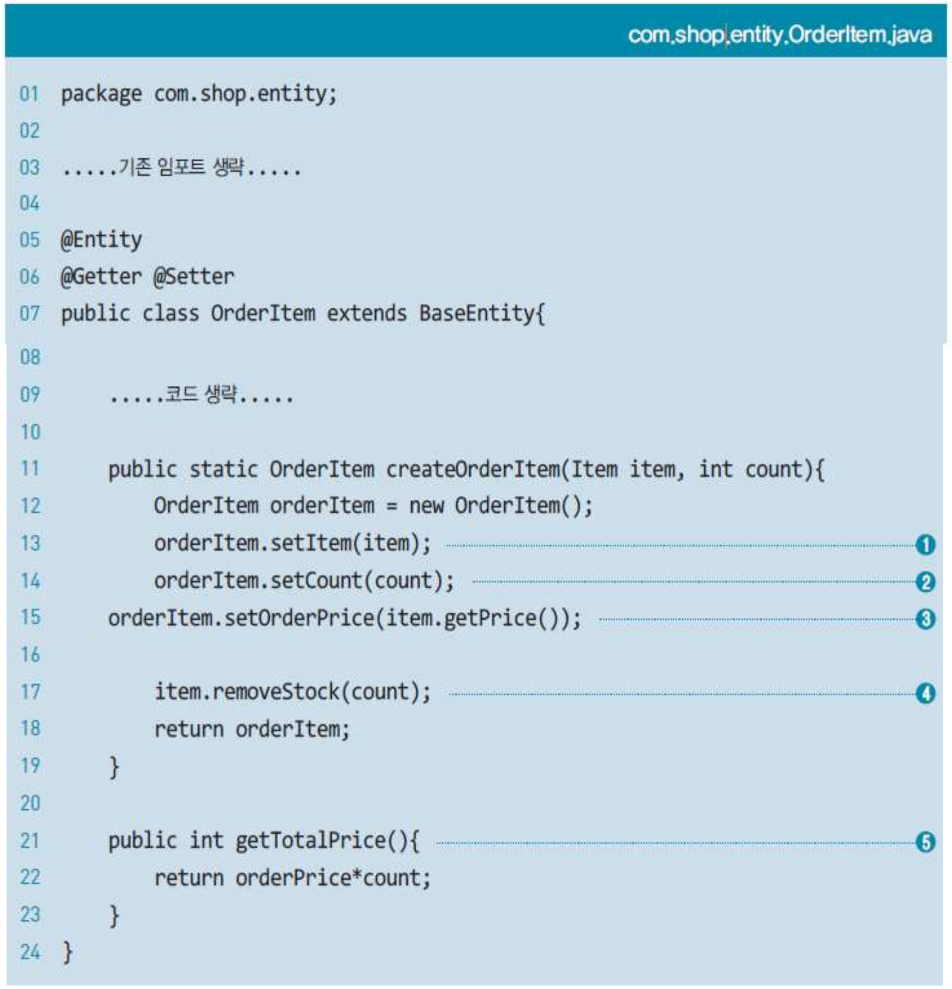


com.shop.entity.OrderItem.java

```
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Getter @Setter
public class OrderItem extends BaseEntity{

    .....코드 생략.....

    public static OrderItem createOrderItem(Item item, int count){
        OrderItem orderItem = new OrderItem();
        orderItem.setItem(item);                                ❶
        orderItem.setCount(count);                              ❷
    orderItem.setOrderPrice(item.getPrice());                   ❸

        item.removeStock(count);                                ❹
        return orderItem;
    }

    public int getTotalPrice(){                                 ❺
        return orderPrice*count;
    }
}
```

7.1 주문 기능 구현

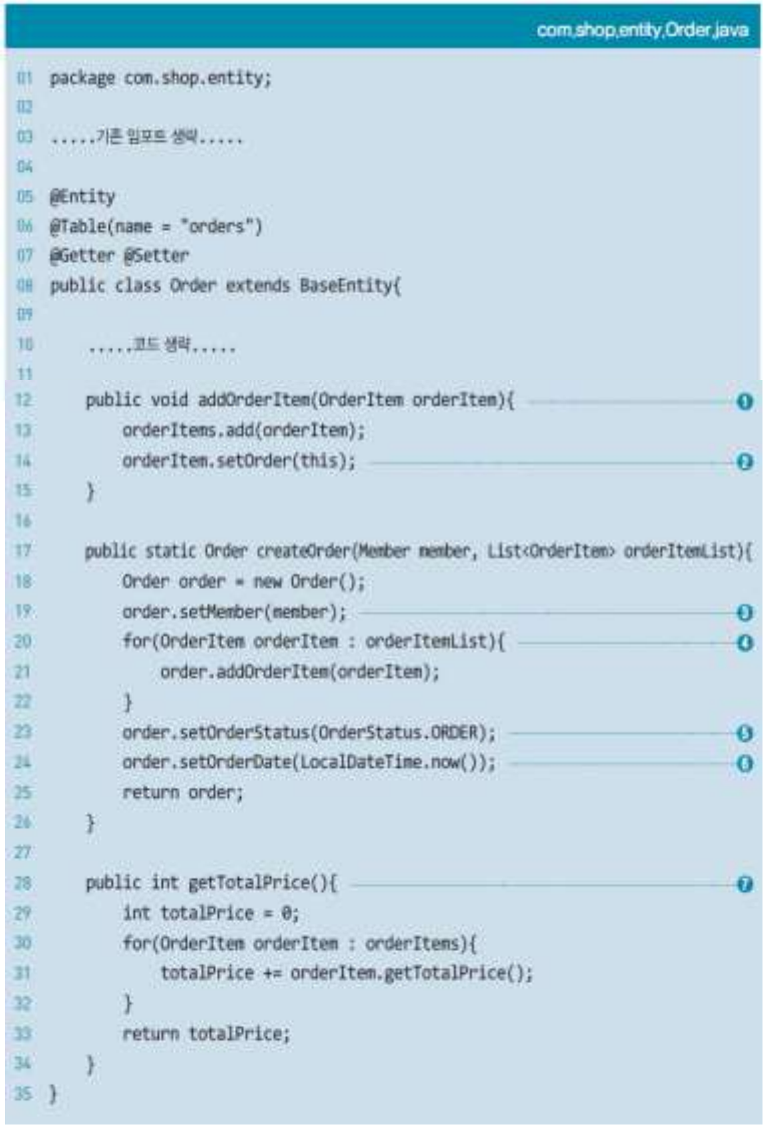
- 주문할 상품과 주문 수량을 통해 OrderItem 객체를 만드는 메소드를 작성



com.shop.entity.OrderItem.java

```
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Getter @Setter
public class OrderItem extends BaseEntity{

    .....코드 생략.....

    public static OrderItem createOrderItem(Item item, int count){
        OrderItem orderItem = new OrderItem();
        orderItem.setItem(item);                          ❶
        orderItem.setCount(count);                        ❷
    orderItem.setOrderPrice(item.getPrice());             ❸

        item.removeStock(count);                          ❹
        return orderItem;
    }

    public int getTotalPrice(){                           ❺
        return orderPrice*count;
    }
}
```

7.1 주문 기능 구현


com.shop.entity.Order.java

```java
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Table(name = "orders")
@Getter @Setter
public class Order extends BaseEntity{

    .....코드 생략.....

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

    public static Order createOrder(Member member, List<OrderItem> orderItemList){
        Order order = new Order();
        order.setMember(member); ❸
        for(OrderItem orderItem : orderItemList){ ❹
            order.addOrderItem(orderItem);
        }
        order.setOrderStatus(OrderStatus.ORDER); ❺
        order.setOrderDate(LocalDateTime.now()); ❻
        return order;
    }

    public int getTotalPrice(){ ❼
        int totalPrice = 0;
        for(OrderItem orderItem : orderItems){
            totalPrice += orderItem.getTotalPrice();
        }
        return totalPrice;
    }
}
```

- 주문 상품 객체를 이용하여 주문 객체를 만드는 메소드 작성

7.1 주문 기능 구현

com.shop.dto.OrderDto.java

```java
package com.shop.dto;

import lombok.Getter;
import lombok.Setter;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

@Getter @Setter
public class OrderDto {

    @NotNull(message = "상품 아이디는 필수 입력 값입니다.")
    private Long itemId;

    @Min(value = 1, message = "최소 주문 수량은 1개 입니다.")
    @Max(value = 999, message = "최대 주문 수량은 999개 입니다.")
    private int count;

}
```

7.1 주문 기능 구현

com.shop.service.OrderService.java

```java
package com.shop.service;

import com.shop.dto.OrderDto;
import com.shop.entity.*;
import com.shop.repository.ItemRepository;
import com.shop.repository.MemberRepository;
import com.shop.repository.OrderRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityNotFoundException;
import java.util.ArrayList;
import java.util.List;
```

7.1 주문 기능 구현

```
@Service
@Transactional
@RequiredArgsConstructor
public class OrderService {

    private final ItemRepository itemRepository;
    private final MemberRepository memberRepository;
    private final OrderRepository orderRepository;

    public Long order(OrderDto orderDto, String email){
        Item item = itemRepository.findById(orderDto.getItemId()) ······ ❶
                .orElseThrow(EntityNotFoundException::new);
        Member member = memberRepository.findByEmail(email); ······ ❷

        List<OrderItem> orderItemList = new ArrayList<>();
        OrderItem orderItem =
OrderItem.createOrderItem(item, orderDto.getCount()); ······ ❸
        orderItemList.add(orderItem);

        Order order = Order.createOrder(member, orderItemList); ······ ❹
        orderRepository.save(order); ······ ❺

        return order.getId();
    }

}
```

7.1 주문 기능 구현

com.shop.controller.OrderController.java

```java
package com.shop.controller;

import com.shop.dto.OrderDto;
import com.shop.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.validation.Valid;
import java.security.Principal;
import java.util.List;

@Controller
@RequiredArgsConstructor
```

7.1 주문 기능 구현

```java
public class OrderController {

    private final OrderService orderService;

    @PostMapping(value = "/order")
    public @ResponseBody ResponseEntity order (@RequestBody @Valid OrderDto orderDto,
          BindingResult bindingResult, Principal principal){ ❶

        if(bindingResult.hasErrors()){ ❷
            StringBuilder sb = new StringBuilder();
            List<FieldError> fieldErrors = bindingResult.getFieldErrors();
            for (FieldError fieldError : fieldErrors) {
                sb.append(fieldError.getDefaultMessage());
            }
            return new ResponseEntity<String>(sb.toString(),
                HttpStatus.BAD_REQUEST); ❸
        }

        String email = principal.getName(); ❹
        Long orderId;

        try {
            orderId = orderService.order(orderDto, email); ❺
        } catch(Exception e){
            return new ResponseEntity<String>(e.getMessage(),
                    HttpStatus.BAD_REQUEST);
        }

        return new ResponseEntity<Long>(orderId, HttpStatus.OK); ❻
    }

}
```

7.1 주문 기능 구현

[함께 해봐요 7-2] 주문 기능 테스트하기

com.shop.service.OrderServiceTest.java

```java
package com.shop.service;

import com.shop.constant.ItemSellStatus;
import com.shop.dto.OrderDto;
import com.shop.entity.Item;
import com.shop.entity.Member;
import com.shop.entity.Order;
import com.shop.entity.OrderItem;
import com.shop.repository.ItemRepository;
import com.shop.repository.MemberRepository;
import com.shop.repository.OrderRepository;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.TestPropertySource;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityNotFoundException;

import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
```

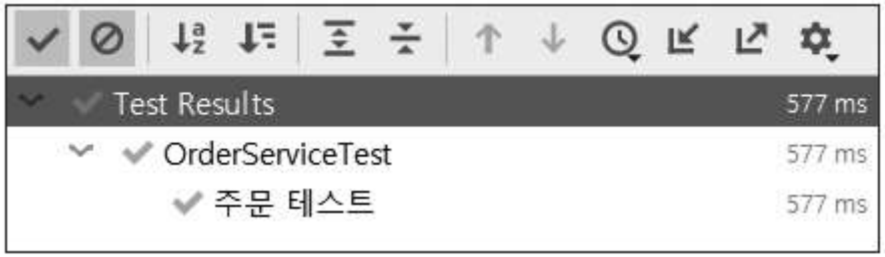
7.1 주문 기능 구현

```java
@SpringBootTest
@Transactional
@TestPropertySource(locations="classpath:application-test.properties")
class OrderServiceTest {

    @Autowired
    private OrderService orderService;

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    ItemRepository itemRepository;

    @Autowired
    MemberRepository memberRepository;

    public Item saveItem(){ ················································ ❶
        Item item = new Item();
        item.setItemNm("테스트 상품");
        item.setPrice(10000);
        item.setItemDetail("테스트 상품 상세 설명");
        item.setItemSellStatus(ItemSellStatus.SELL);
        item.setStockNumber(100);
        return itemRepository.save(item);
    }
```

7.1 주문 기능 구현

```java
    public Member saveMember(){                                          ❷
        Member member = new Member();
        member.setEmail("test@test.com");
        return memberRepository.save(member);
    }

    @Test
    @DisplayName("주문 테스트")
    public void order(){
        Item item = saveItem();
        Member member = saveMember();

        OrderDto orderDto = new OrderDto();
        orderDto.setCount(10);                                           ❸
        orderDto.setItemId(item.getId());                                ❹

        Long orderId = orderService.order(orderDto, member.getEmail());  ❺

        Order order = orderRepository.findById(orderId)                  ❻
                .orElseThrow(EntityNotFoundException::new);

        List<OrderItem> orderItems = order.getOrderItems();

        int totalPrice = orderDto.getCount()*item.getPrice();            ❼

        assertEquals(totalPrice, order.getTotalPrice());                 ❽
    }

}
```

7.1 주문 기능 구현

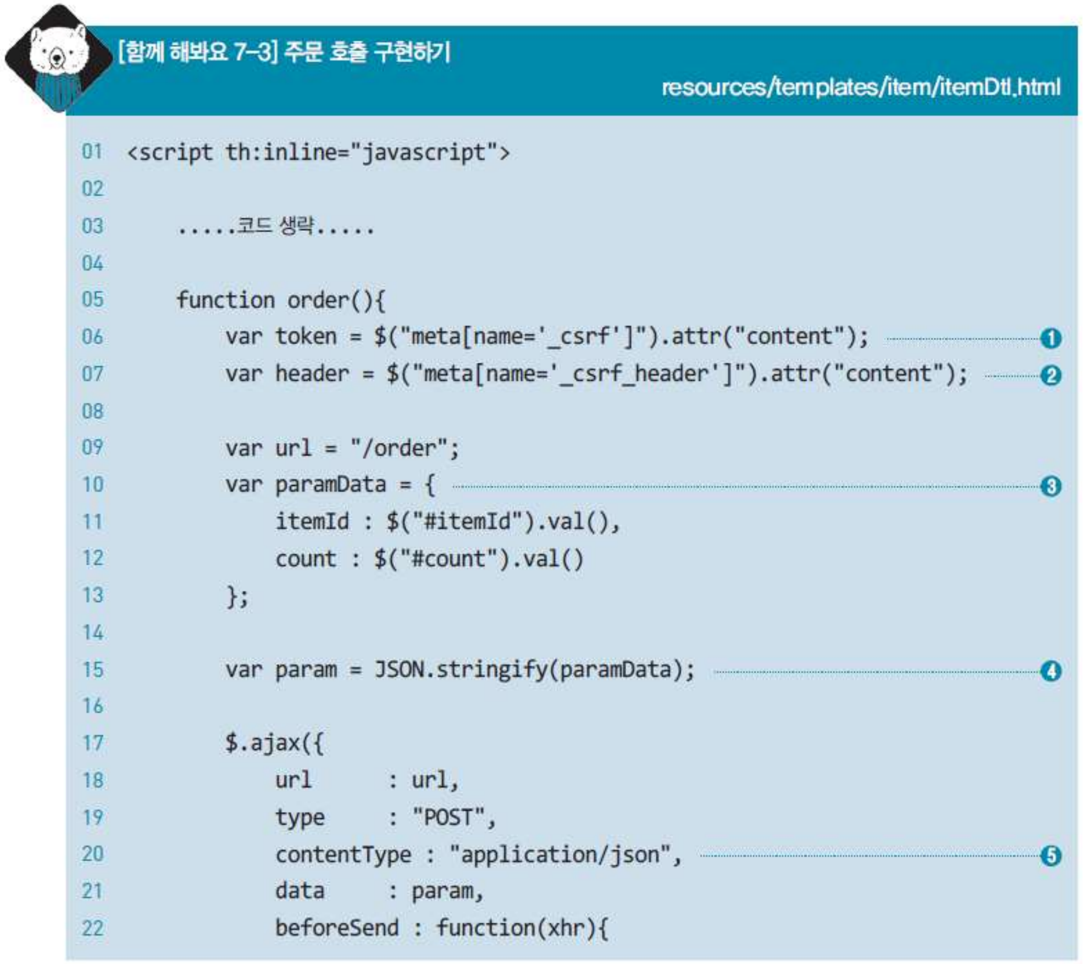


```
Test Results              577 ms
  OrderServiceTest        577 ms
    주문 테스트             577 ms
```

[그림 7-1] 주문 테스트 코드 실행 결과

7.1 주문 기능 구현

- 상품 상세 페이지에서 구현한 주문 로직을 호출하는 코드 작성


[함께 해봐요 7-3] 주문 호출 구현하기

resources/templates/item/itemDtl.html

```
<script th:inline="javascript">

    .....코드 생략.....

    function order(){
        var token = $("meta[name='_csrf']").attr("content"); ❶
        var header = $("meta[name='_csrf_header']").attr("content"); ❷

        var url = "/order";
        var paramData = { ❸
            itemId : $("#itemId").val(),
            count : $("#count").val()
        };

        var param = JSON.stringify(paramData); ❹

        $.ajax({
            url      : url,
            type     : "POST",
            contentType : "application/json", ❺
            data     : param,
            beforeSend : function(xhr){
```

7.1 주문 기능 구현

```
                /* 데이터를 전송하기 전에 헤더에 csrf 값을 설정 */
                xhr.setRequestHeader(header, token);
            },
            dataType : "json",                                    ❻
            cache    : false,
            success  : function(result, status){                  ❼
                alert("주문이 완료 되었습니다.");
                location.href='/';
            },
            error : function(jqXHR, status, error){

                if(jqXHR.status == '401'){                        ❽
                    alert('로그인 후 이용해주세요');
                    location.href='/members/login';
                } else{
                    alert(jqXHR.responseText);                    ❾
                }

            }
        });
    }
</script>
```

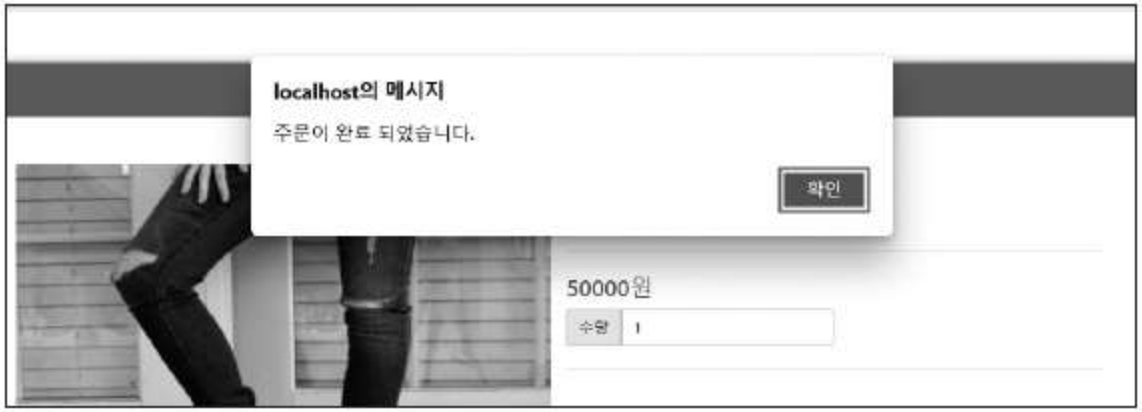
7.1 주문 기능 구현

- 주문하기 버튼 클릭 시 order() 함수 호출 로직 추가

resources/templates/item/itemDtl.html

```
<button type="button" class="btn btn-primary btn-lg" onclick="order()">주문하기
</button>
```

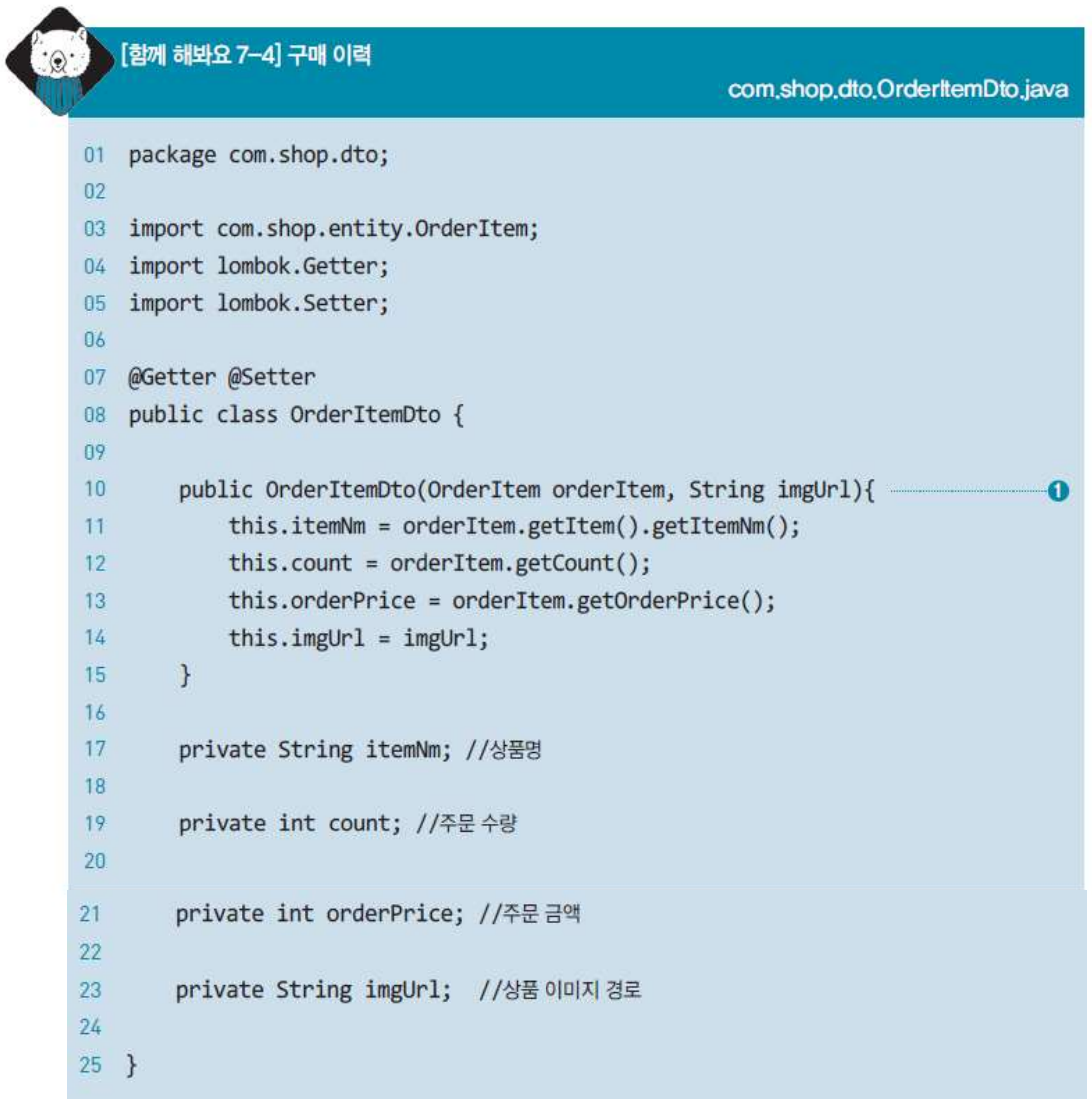
7.1 주문 기능 구현


localhost의 메시지

주문이 완료 되었습니다.

[그림 7-2] 상품 주문 성공

7.2 주문 이력 조회

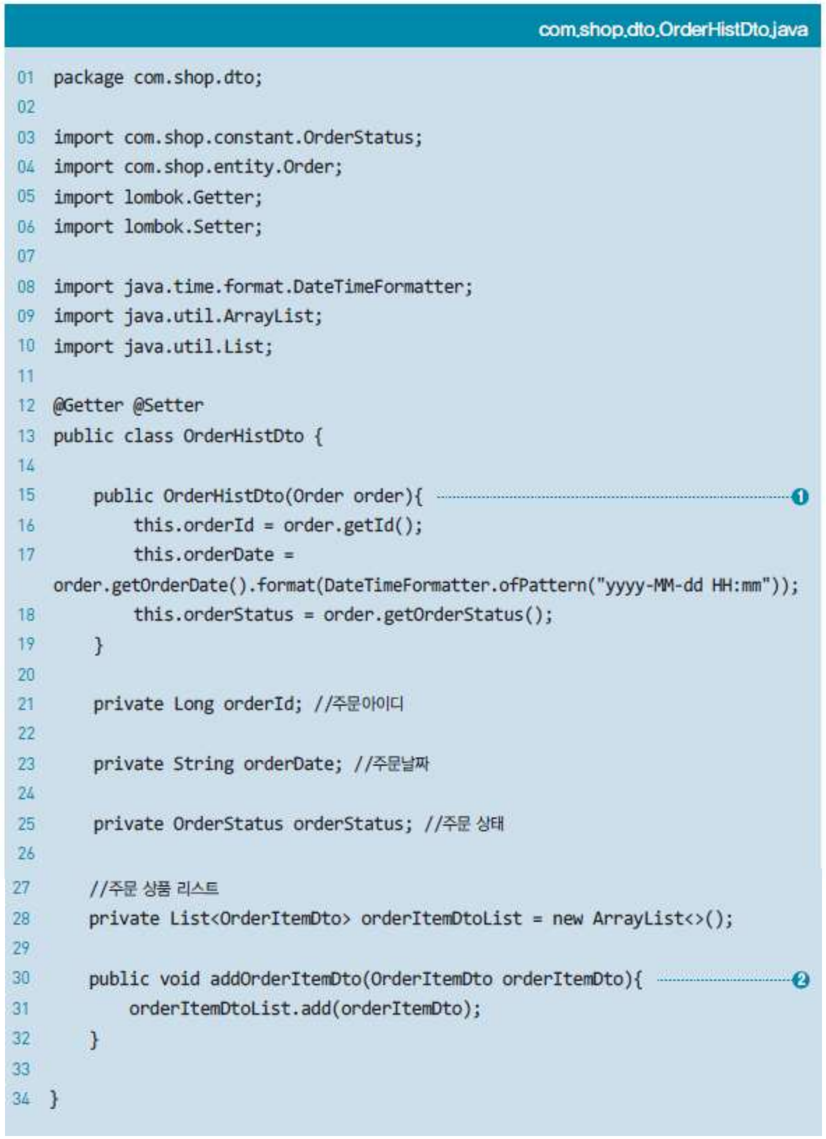


[함께 해봐요 7-4] 구매 이력

com.shop.dto.OrderItemDto.java

```java
package com.shop.dto;

import com.shop.entity.OrderItem;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class OrderItemDto {

    public OrderItemDto(OrderItem orderItem, String imgUrl){ ❶
        this.itemNm = orderItem.getItem().getItemNm();
        this.count = orderItem.getCount();
        this.orderPrice = orderItem.getOrderPrice();
        this.imgUrl = imgUrl;
    }

    private String itemNm; //상품명

    private int count; //주문 수량

    private int orderPrice; //주문 금액

    private String imgUrl;   //상품 이미지 경로

}
```

7.2 주문 이력 조회

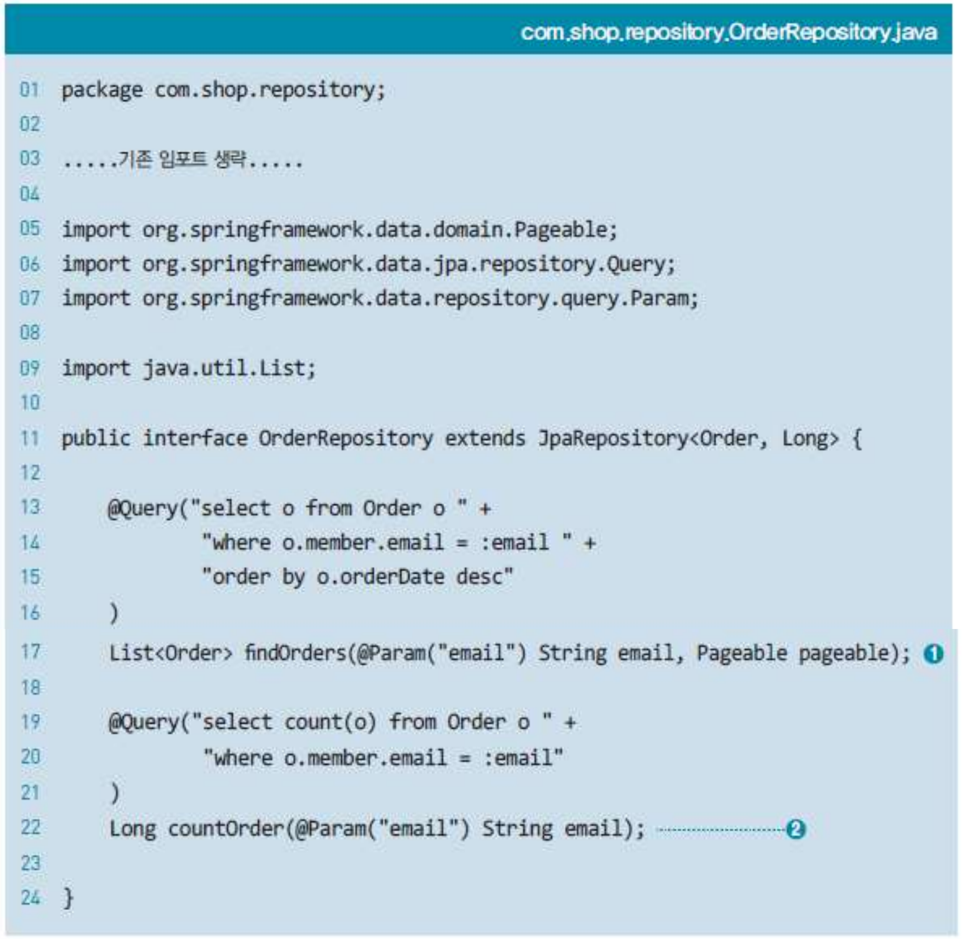


com.shop.dto.OrderHistDto.java

```java
package com.shop.dto;

import com.shop.constant.OrderStatus;
import com.shop.entity.Order;
import lombok.Getter;
import lombok.Setter;

import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

@Getter @Setter
public class OrderHistDto {

    public OrderHistDto(Order order){ ········································ ❶
        this.orderId = order.getId();
        this.orderDate =
order.getOrderDate().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"));
        this.orderStatus = order.getOrderStatus();
    }

    private Long orderId; //주문아이디

    private String orderDate; //주문날짜

    private OrderStatus orderStatus; //주문 상태

    //주문 상품 리스트
    private List<OrderItemDto> orderItemDtoList = new ArrayList<>();

    public void addOrderItemDto(OrderItemDto orderItemDto){ ···················· ❷
        orderItemDtoList.add(orderItemDto);
    }

}
```

7.2 주문 이력 조회

- 주문이력 조회 쿼리 작성



com.shop.repository.OrderRepository.java

```java
package com.shop.repository;

.....기존 임포트 생략.....

import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface OrderRepository extends JpaRepository<Order, Long> {

    @Query("select o from Order o " +
            "where o.member.email = :email " +
            "order by o.orderDate desc"
    )
    List<Order> findOrders(@Param("email") String email, Pageable pageable); ❶

    @Query("select count(o) from Order o " +
            "where o.member.email = :email"
    )
    Long countOrder(@Param("email") String email); ········❷

}
```

7.2 주문 이력 조회

- 주문 상품의 대표 이미지를 보여주기 위한 쿼리 작성

com.shop.repository.ItemImgRepository.java

```java
package com.shop.repository;

.....기존 임포트 생략.....

public interface ItemImgRepository extends JpaRepository<ItemImg, Long> {

    List<ItemImg> findByItemIdOrderByIdAsc(Long itemId);

    ItemImg findByItemIdAndRepimgYn(Long itemId, String repimgYn);
}
```

7.2 주문 이력 조회

com.shop.service.OrderService.java

```java
package com.shop.service;

.....기존 임포트 생략.....

import com.shop.dto.OrderHistDto;
import com.shop.dto.OrderItemDto;
import com.shop.repository.ItemImgRepository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;

@Service
@Transactional
@RequiredArgsConstructor
public class OrderService {

    private final ItemRepository itemRepository;
    private final MemberRepository memberRepository;
    private final OrderRepository orderRepository;
    private final ItemImgRepository itemImgRepository;

    .....코드 생략.....

```

7.2 주문 이력 조회

```java
    @Transactional(readOnly = true)
    public Page<OrderHistDto> getOrderList(String email, Pageable pageable) {

        List<Order> orders  = orderRepository.findOrders(email, pageable); ❶
        Long totalCount = orderRepository.countOrder(email); ❷

        List<OrderHistDto> orderHistDtos = new ArrayList<>();

        for (Order order : orders) { ❸
            OrderHistDto orderHistDto = new OrderHistDto(order);
            List<OrderItem> orderItems = order.getOrderItems();
            for (OrderItem orderItem : orderItems) {
                ItemImg itemImg = itemImgRepository.findByItemIdAndRepimgYn
(orderItem.getItem().getId(), "Y"); ❹
                OrderItemDto orderItemDto =
new OrderItemDto(orderItem, itemImg.getImgUrl());
                orderHistDto.addOrderItemDto(orderItemDto);
            }

            orderHistDtos.add(orderHistDto);
        }

        return new PageImpl<OrderHistDto>(orderHistDtos, pageable, totalCount); ❺
    }

}
```

7.2 주문 이력 조회

com.shop.controller.OrderController.java

```java
package com.shop.controller;

.....기존 임포트 생략.....

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import com.shop.dto.OrderHistDto;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.ui.Model;
import java.util.Optional;

@Controller
@RequiredArgsConstructor
public class OrderController {

    .....코드 생략.....

    @GetMapping(value = {"/orders", "/orders/{page}"})
    public String orderHist(@PathVariable("page") Optional<Integer> page,
Principal principal, Model model){
        Pageable pageable = PageRequest.of(page.isPresent() ? page.get() : 0, 4); ❶

        Page<OrderHistDto> ordersHistDtoList =
orderService.getOrderList(principal.getName(), pageable); ······························ ❷

        model.addAttribute("orders", ordersHistDtoList);
        model.addAttribute("page", pageable.getPageNumber());
        model.addAttribute("maxPage", 5);
        return "order/orderHist";
    }

}
```

7.2 주문 이력 조회

- 주문 이력 페이지 orderHist.html 작성

resources/templates/order/orderHist.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layouts/layout1}">

<head>
    <meta name="_csrf" th:content="${_csrf.token}"/>
    <meta name="_csrf_header" th:content="${_csrf.headerName}"/>
</head>

<!-- 사용자 CSS 추가 -->
<th:block layout:fragment="css">
    <style>
        .content-mg{
            margin-left:30%;
            margin-right:30%;
            margin-top:2%;
            margin-bottom:100px;
        }
        .repImgDiv{
            margin-right:15px;
            margin-left:15px;
            height:auto;
        }
        .repImg{
            height:100px;
            width:100px;
        }
        .card{
            width:750px;
            height:100%;
            padding:30px;
            margin-bottom:20px;
        }
        .fs18{
            font-size:18px
        }
        .fs24{
            font-size:24px
        }
    </style>
</th:block>
```

7.2 주문 이력 조회

```
<div layout:fragment="content" class="content-mg">

    <h2 class="mb-4">
        구매 이력
    </h2>

    <div th:each="order : ${orders.getContent()}">

        <div class="d-flex mb-3 align-self-center">
            <h4 th:text="${order.orderDate} + ' 주문'"></h4>
            <div class="ml-3">
                <th:block th:if="${order.orderStatus ==
T(com.shop.constant.OrderStatus).ORDER}">
                    <button type="button" class="btn btn-outline-secondary"
th:value="${order.orderId}">주문취소</button>
                </th:block>
                <th:block th:unless="${order.orderStatus ==
T(com.shop.constant.OrderStatus).ORDER}">
                    <h4>(취소 완료)</h4>
                </th:block>
            </div>
        </div>
        <div class="card d-flex">
```

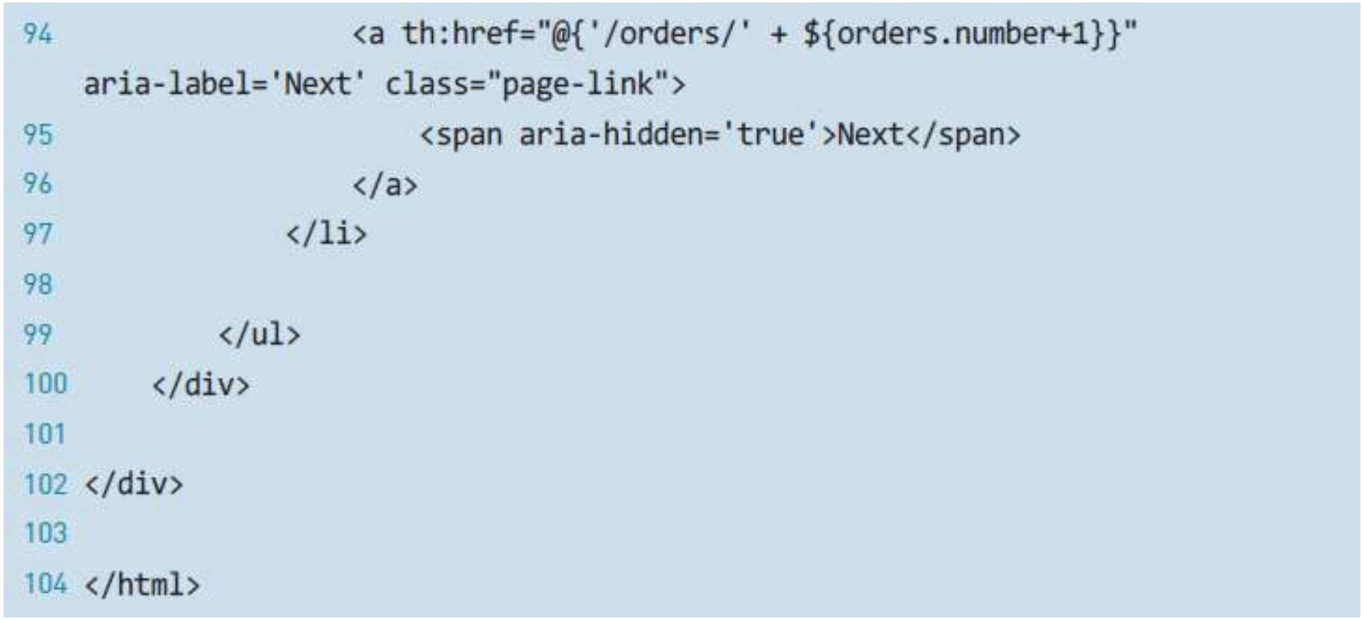
7.2 주문 이력 조회

```
            <div th:each="orderItem : ${order.orderItemDtoList}"
class="d-flex mb-3">
                <div class="repImgDiv">
                    <img th:src="${orderItem.imgUrl}"
class = "rounded repImg" th:alt="${orderItem.itemNm}">
                </div>
                <div class="align-self-center w-75">
                    <span th:text="${orderItem.itemNm}"
class="fs24 font-weight-bold"></span>
                    <div class="fs18 font-weight-light">
                        <span th:text="${orderItem.orderPrice} +'원'">
                        </span>
                        <span th:text="${orderItem.count} +'개'"></span>
                    </div>
                </div>
            </div>
        </div>

    </div>

```

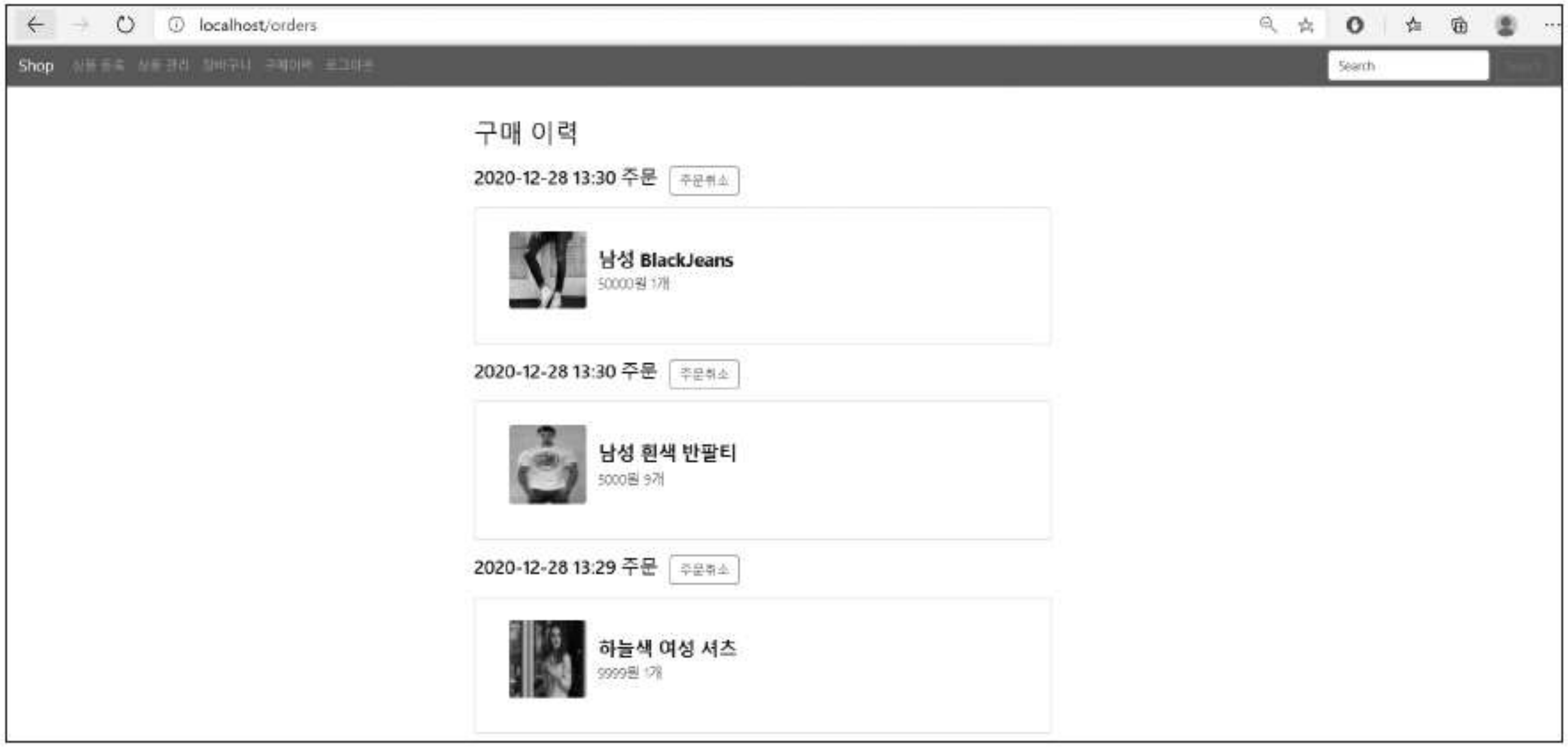
7.2 주문 이력 조회

```
    <div th:with="start=${(orders.number/maxPage)*maxPage + 1},
end=(${(orders.totalPages == 0) ? 1 : (start + (maxPage - 1)
< orders.totalPages ? start + (maxPage - 1) : orders.totalPages)})" >
        <ul class="pagination justify-content-center">

            <li class="page-item"
th:classappend="${orders.number eq 0}?'disabled':''">
                <a th:href="@{'/orders/' + ${orders.number-1}}"
aria-label='Previous' class="page-link">
                    <span aria-hidden='true'>Previous</span>
                </a>
            </li>

            <li class="page-item"
th:each="page: ${#numbers.sequence(start, end)}"
th:classappend="${orders.number eq page-1}?'active':''">
                <a th:href="@{'/orders/' + ${page-1}}" th:inline="text"
class="page-link">[[${page}]]</a>
            </li>

            <li class="page-item"
th:classappend="${orders.number+1 ge orders.totalPages}?'disabled':''">
```

7.2 주문 이력 조회


```
                <a th:href="@{'/orders/' + ${orders.number+1}}" aria-label='Next' class="page-link">
                    <span aria-hidden='true'>Next</span>
                </a>
            </li>

        </ul>
    </div>

</div>

</html>
```

7.2 주문 이력 조회



구매 이력

2020-12-28 13:30 주문 주문취소

남성 BlackJeans
50000원 1개

2020-12-28 13:30 주문 주문취소

남성 흰색 반팔티
5000원 9개

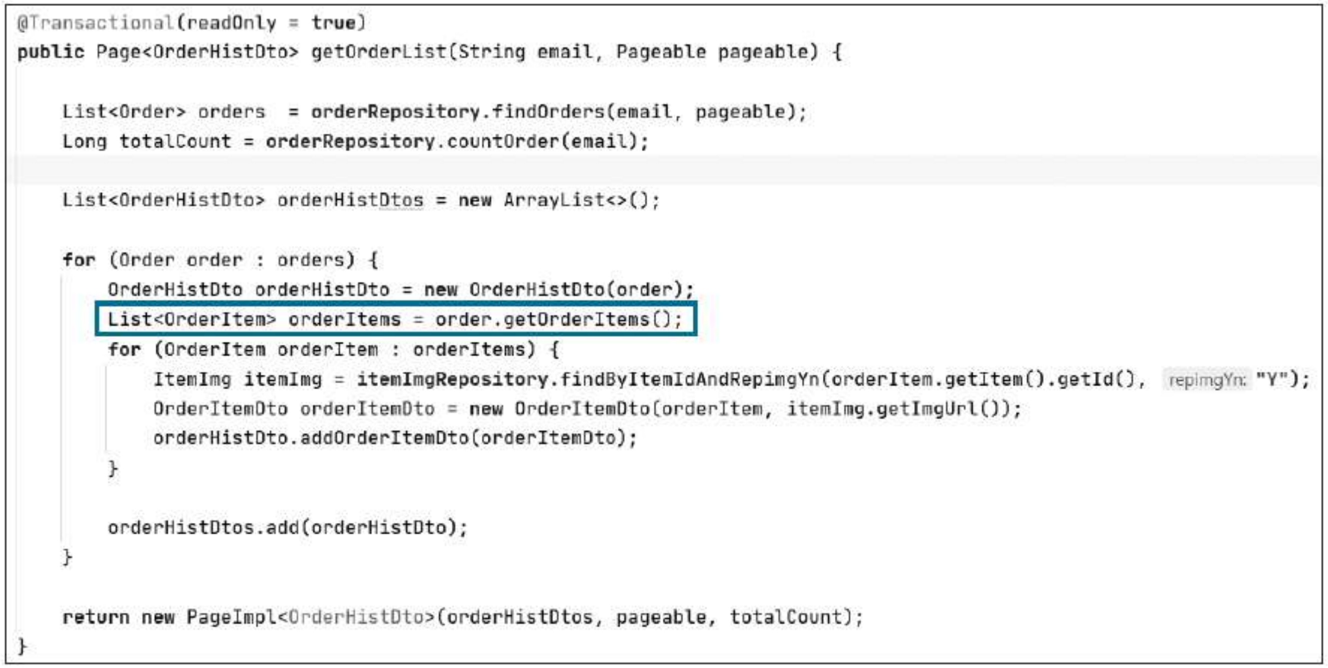
2020-12-28 13:29 주문 주문취소

하늘색 여성 셔츠
9999원 1개

[그림 7-3] 주문 이력 페이지

7.2 주문 이력 조회

- OrderService 클래스에 구현한getOrderList() 메소드에서 for문을 순회하면서 order.getOrderItems()를 호출할 때마다 조회 쿼리문이 추가로 실행

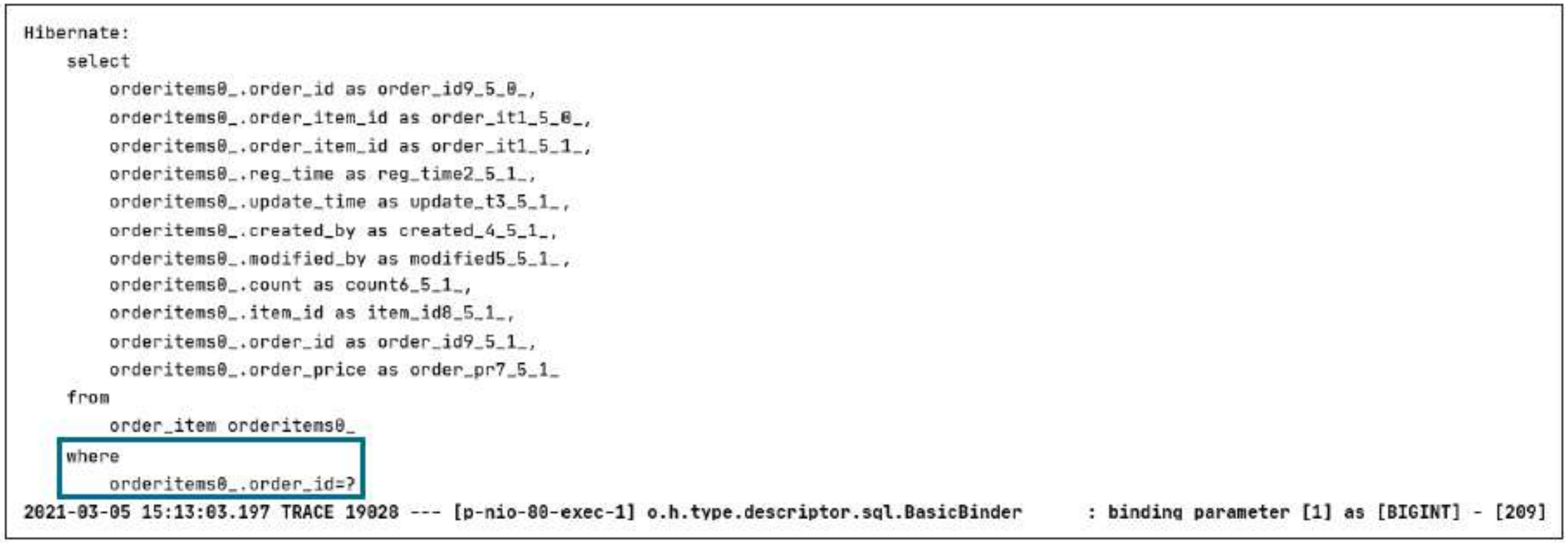


```java
@Transactional(readOnly = true)
public Page<OrderHistDto> getOrderList(String email, Pageable pageable) {

    List<Order> orders  = orderRepository.findOrders(email, pageable);
    Long totalCount = orderRepository.countOrder(email);

    List<OrderHistDto> orderHistDtos = new ArrayList<>();

    for (Order order : orders) {
        OrderHistDto orderHistDto = new OrderHistDto(order);
        List<OrderItem> orderItems = order.getOrderItems();
        for (OrderItem orderItem : orderItems) {
            ItemImg itemImg = itemImgRepository.findByItemIdAndRepimgYn(orderItem.getItem().getId(),  repimgYn: "Y");
            OrderItemDto orderItemDto = new OrderItemDto(orderItem, itemImg.getImgUrl());
            orderHistDto.addOrderItemDto(orderItemDto);
        }

        orderHistDtos.add(orderHistDto);
    }

    return new PageImpl<OrderHistDto>(orderHistDtos, pageable, totalCount);
}
```

[그림 7-4] 주문 리스트 조회 로직

7.2 주문 이력 조회

- OrderService 클래스에 구현한getOrderList() 메소드에서 for문을 순회하면서 order.getOrderItems()를 호출할 때마다 조회 쿼리문이 추가로 실행

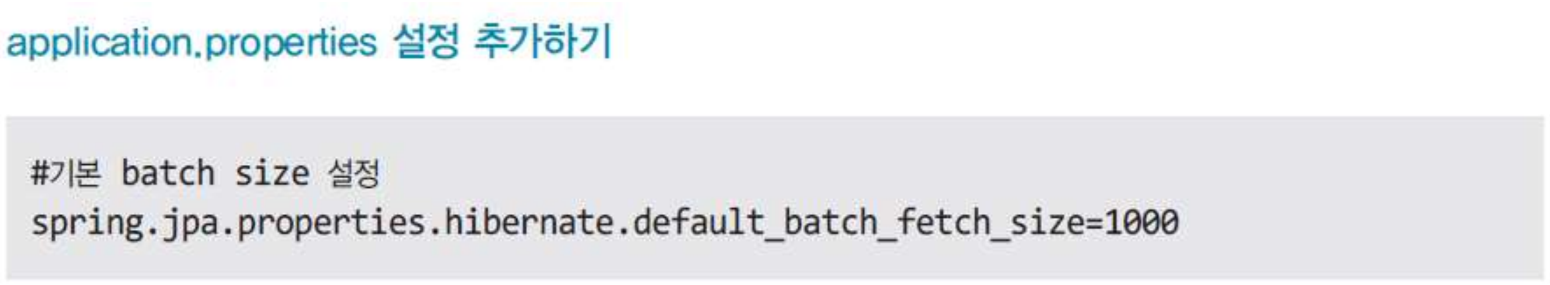


```
Hibernate:
    select
        orderitems0_.order_id as order_id9_5_0_,
        orderitems0_.order_item_id as order_it1_5_0_,
        orderitems0_.order_item_id as order_it1_5_1_,
        orderitems0_.reg_time as reg_time2_5_1_,
        orderitems0_.update_time as update_t3_5_1_,
        orderitems0_.created_by as created_4_5_1_,
        orderitems0_.modified_by as modified5_5_1_,
        orderitems0_.count as count6_5_1_,
        orderitems0_.item_id as item_id8_5_1_,
        orderitems0_.order_id as order_id9_5_1_,
        orderitems0_.order_price as order_pr7_5_1_
    from
        order_item orderitems0_
    where
        orderitems0_.order_id=?
2021-03-05 15:13:03.197 TRACE 19028 --- [p-nio-80-exec-1] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [BIGINT] - [209]
```

[그림 7-5] 주문 상품 리스트 조회 쿼리

7.2 주문 이력 조회

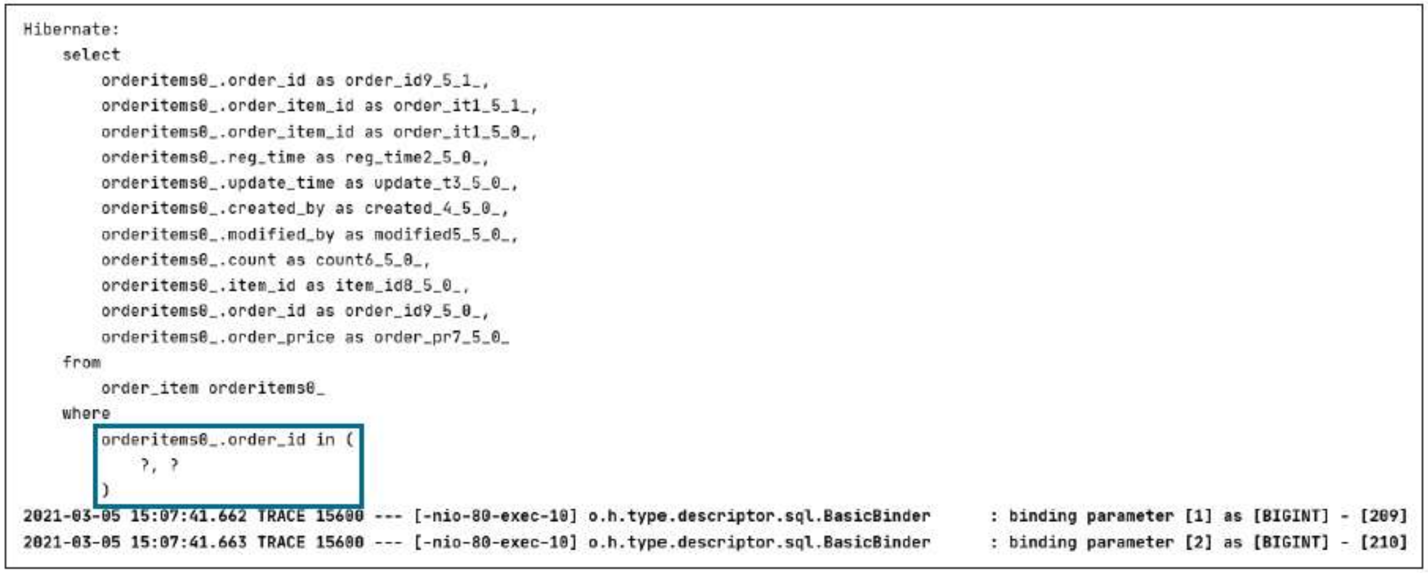
- orders의 주문 아이디를 “where order_id in (209, 210, 211, 212)” 이런 식으로 in 쿼리로 한번에 조회할 수 있다면 100개가 실행될 쿼리를 하나의 쿼리로 조회가능
- “default_batch_fetch_size”라는 옵션을 사용하여 조회 쿼리 한 번으로 지정한 사이즈 만큼 한 번에 조회 가능


application.properties 설정 추가하기

```
#기본 batch size 설정
spring.jpa.properties.hibernate.default_batch_fetch_size=1000
```

7.2 주문 이력 조회

- 옵션 추가 후 조건절에 in 쿼리문이 실행되는 것을 볼 수 있음
- JPA 사용 시 N+1 문제를 많이 만나게 되는데 성능상 이슈가 생길 수 있기 때문에 조심해서 사용 필요

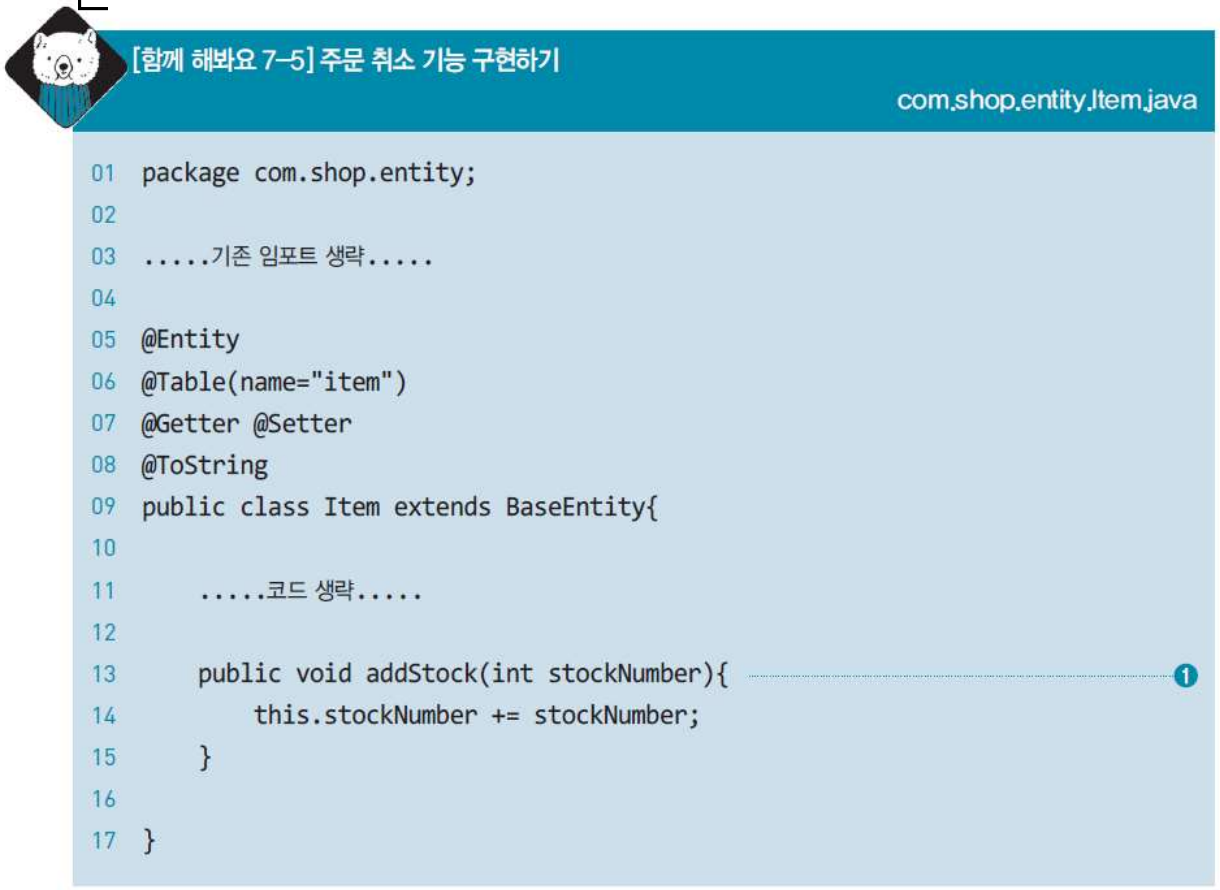


```
Hibernate:
    select
        orderitems0_.order_id as order_id9_5_1_,
        orderitems0_.order_item_id as order_it1_5_1_,
        orderitems0_.order_item_id as order_it1_5_0_,
        orderitems0_.reg_time as reg_time2_5_0_,
        orderitems0_.update_time as update_t3_5_0_,
        orderitems0_.created_by as created_4_5_0_,
        orderitems0_.modified_by as modified5_5_0_,
        orderitems0_.count as count6_5_0_,
        orderitems0_.item_id as item_id8_5_0_,
        orderitems0_.order_id as order_id9_5_0_,
        orderitems0_.order_price as order_pr7_5_0_
    from
        order_item orderitems0_
    where
        orderitems0_.order_id in (
            ?, ?
        )
2021-03-05 15:07:41.662 TRACE 15600 --- [-nio-80-exec-10] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [BIGINT] - [209]
2021-03-05 15:07:41.663 TRACE 15600 --- [-nio-80-exec-10] o.h.type.descriptor.sql.BasicBinder      : binding parameter [2] as [BIGINT] - [210]
```

[그림 7-6] default_batch_fetch_size 적용 후 주문 상품 리스트 조회 쿼리

7.3 주문 취소

- 주문할 때 상품의 재고를 감소시켰던 만큼 다시 더해주는 addStock 메소드 구현



[함께 해봐요 7-5] 주문 취소 기능 구현하기

com.shop.entity.Item.java

```
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Table(name="item")
@Getter @Setter
@ToString
public class Item extends BaseEntity{

    .....코드 생략.....

    public void addStock(int stockNumber){ ❶
        this.stockNumber += stockNumber;
    }

}
```

7.3 주문 취소

com.shop.entity.OrderItem.java

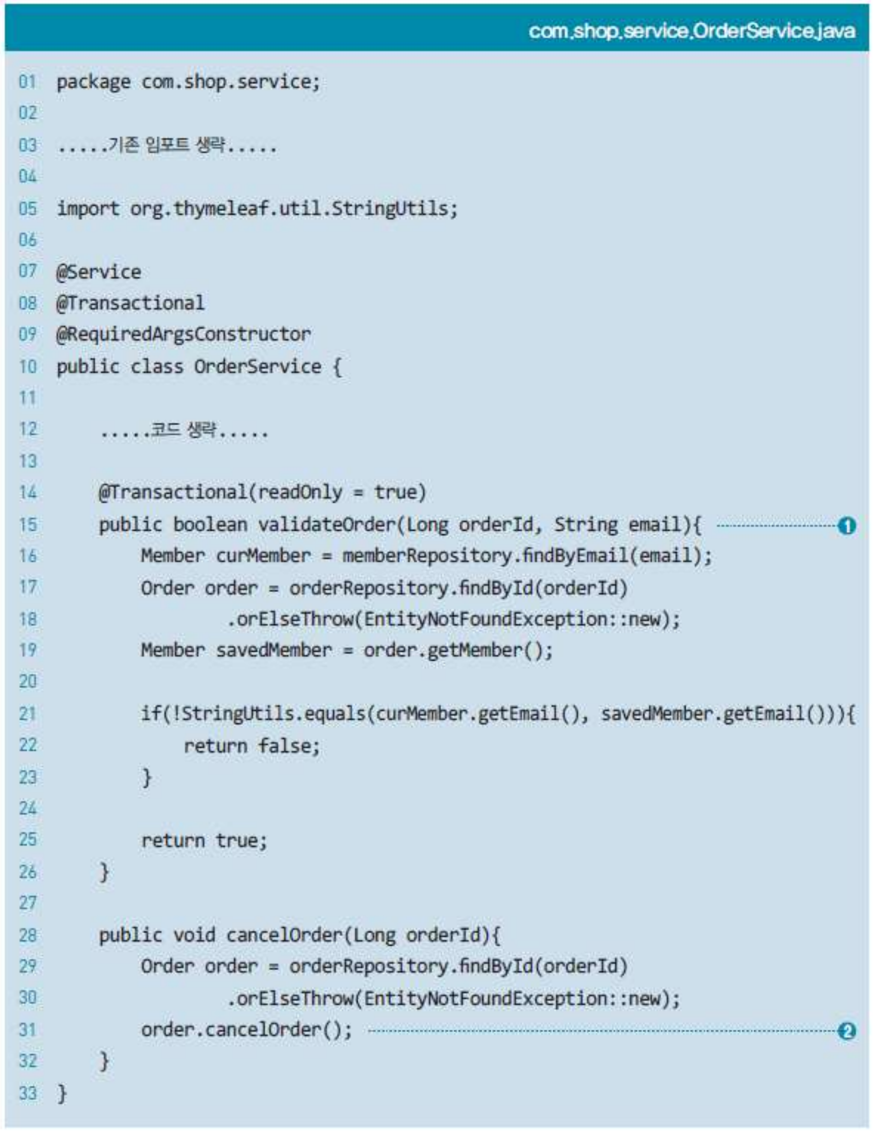
```java
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Getter @Setter
public class OrderItem extends BaseEntity{

    .....코드 생략.....

    public void cancel() { ................................................ ①
        this.getItem().addStock(count);
    }

}
```

7.3 주문 취소

com.shop.entity.Order.java

```java
package com.shop.entity;

.....기존 임포트 생략.....

@Entity
@Table(name = "orders")
@Getter @Setter
public class Order extends BaseEntity{

    .....코드 생략.....

    public void cancelOrder(){
        this.orderStatus = OrderStatus.CANCEL;

        for(OrderItem orderItem : orderItems){
            orderItem.cancel();
        }
    }

}
```

7.3 주문 취소



com.shop.service.OrderService.java

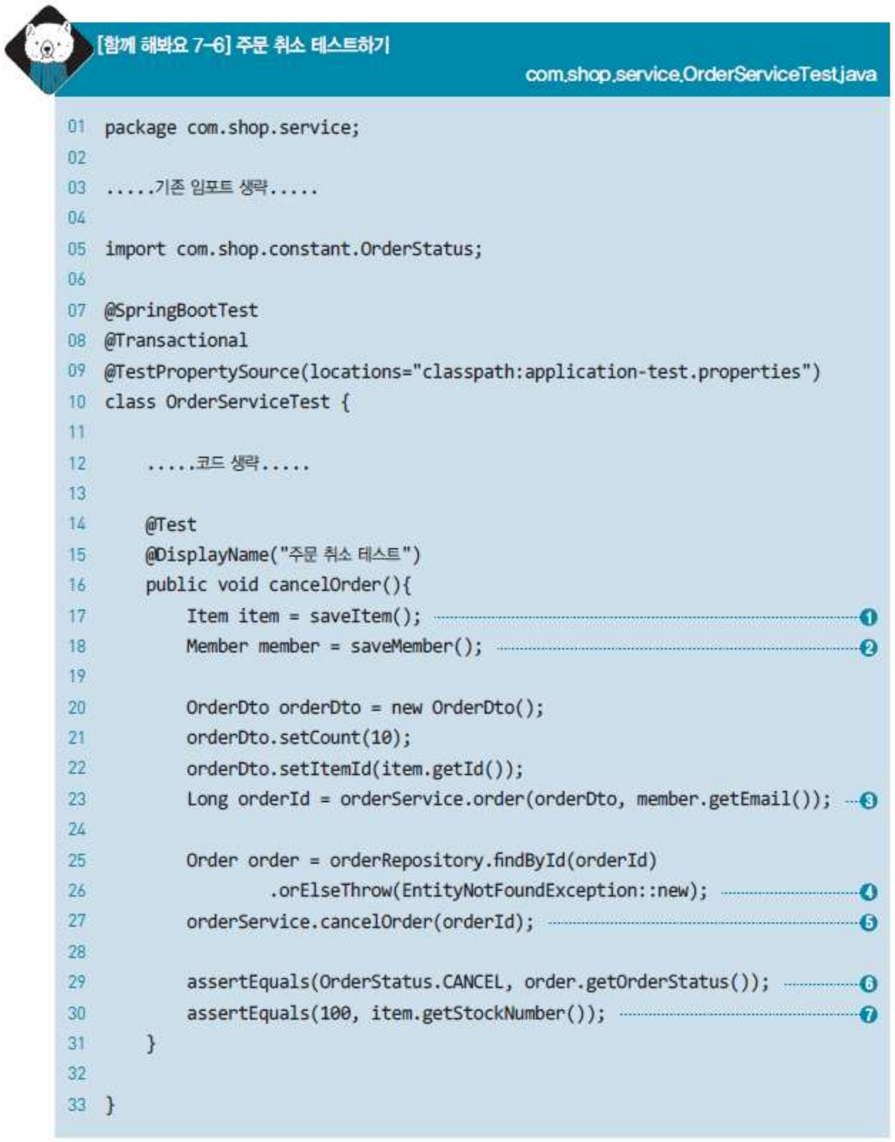
```java
package com.shop.service;

.....기존 임포트 생략.....

import org.thymeleaf.util.StringUtils;

@Service
@Transactional
@RequiredArgsConstructor
public class OrderService {

    .....코드 생략.....

    @Transactional(readOnly = true)
    public boolean validateOrder(Long orderId, String email){ ········ ❶
        Member curMember = memberRepository.findByEmail(email);
        Order order = orderRepository.findById(orderId)
                .orElseThrow(EntityNotFoundException::new);
        Member savedMember = order.getMember();

        if(!StringUtils.equals(curMember.getEmail(), savedMember.getEmail())){
            return false;
        }

        return true;
    }

    public void cancelOrder(Long orderId){
        Order order = orderRepository.findById(orderId)
                .orElseThrow(EntityNotFoundException::new);
        order.cancelOrder(); ········ ❷
    }
}
```

7.3 주문 취소

com.shop.controller.OrderController.java

```
package com.shop.controller;

.....기존 임포트 생략.....

@Controller
@RequiredArgsConstructor
public class OrderController {

    .....코드 생략.....

    @PostMapping("/order/{orderId}/cancel")
    public @ResponseBody ResponseEntity cancelOrder(@PathVariable("orderId") Long orderId , Principal principal){

        if(!orderService.validateOrder(orderId, principal.getName())){ ❶
            return new ResponseEntity<String>("주문 취소 권한이 없습니다.", HttpStatus.FORBIDDEN);
        }

        orderService.cancelOrder(orderId); ❷
        return new ResponseEntity<Long>(orderId, HttpStatus.OK);
    }

}
```

7.3 주문 취소

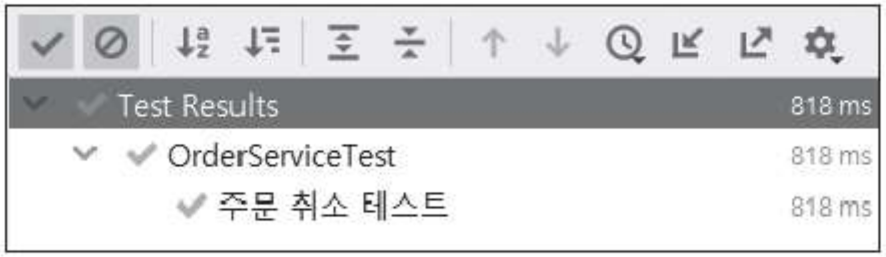


[함께 해봐요 7-6] 주문 취소 테스트하기

com.shop.service.OrderServiceTest.java

```
package com.shop.service;

.....기존 임포트 생략.....

import com.shop.constant.OrderStatus;

@SpringBootTest
@Transactional
@TestPropertySource(locations="classpath:application-test.properties")
class OrderServiceTest {

    .....코드 생략.....

    @Test
    @DisplayName("주문 취소 테스트")
    public void cancelOrder(){
        Item item = saveItem();                                          ❶
        Member member = saveMember();                                    ❷

        OrderDto orderDto = new OrderDto();
        orderDto.setCount(10);
        orderDto.setItemId(item.getId());
        Long orderId = orderService.order(orderDto, member.getEmail());  ❸

        Order order = orderRepository.findById(orderId)
                .orElseThrow(EntityNotFoundException::new);              ❹
        orderService.cancelOrder(orderId);                               ❺

        assertEquals(OrderStatus.CANCEL, order.getOrderStatus());        ❻
        assertEquals(100, item.getStockNumber());                        ❼
    }

}
```

7.3 주문 취소

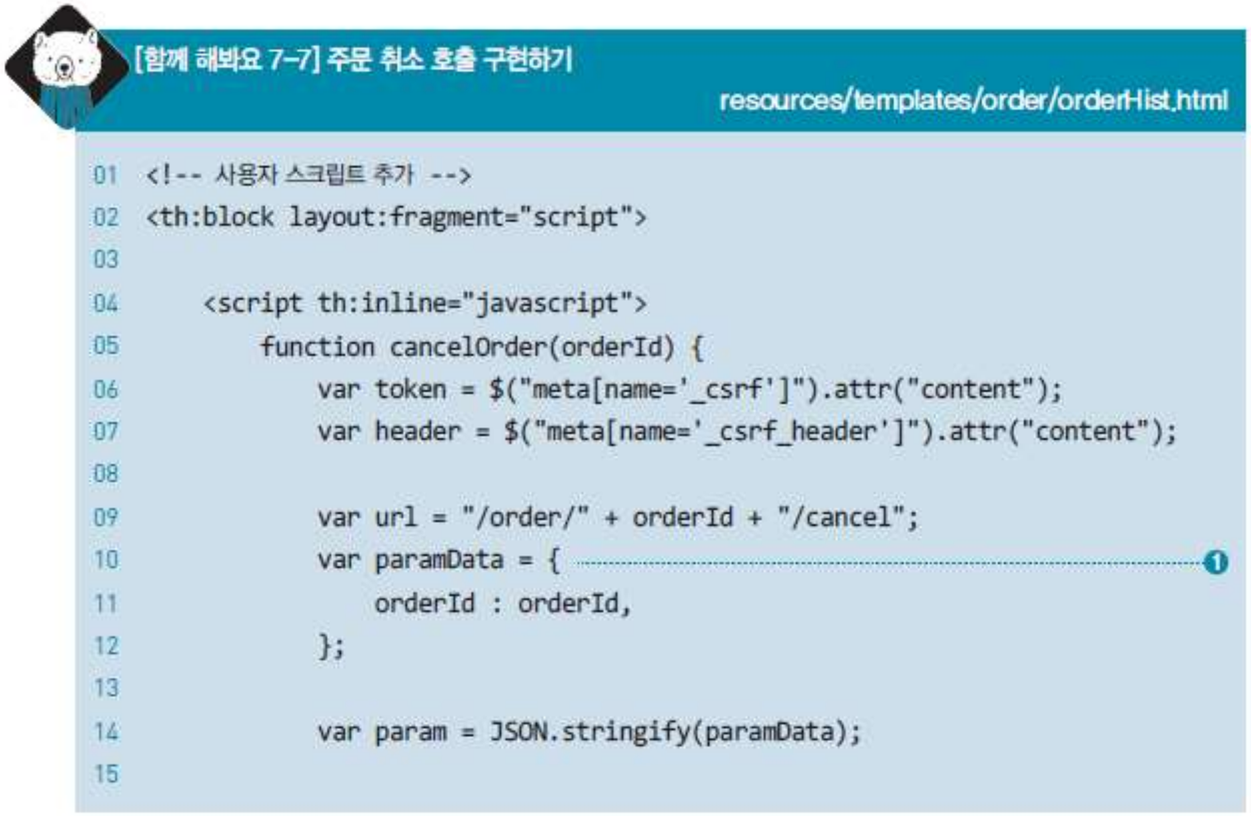


```
Test Results                818 ms
  OrderServiceTest          818 ms
    주문 취소 테스트          818 ms
```

[그림 7-7] 주문 취소 테스트 실행 결과

7.3 주문 취소

- 주문 취소 기능을 호출하는 자바스크립트 함수 구현



[함께 해봐요 7-7] 주문 취소 호출 구현하기

resources/templates/order/orderHist.html

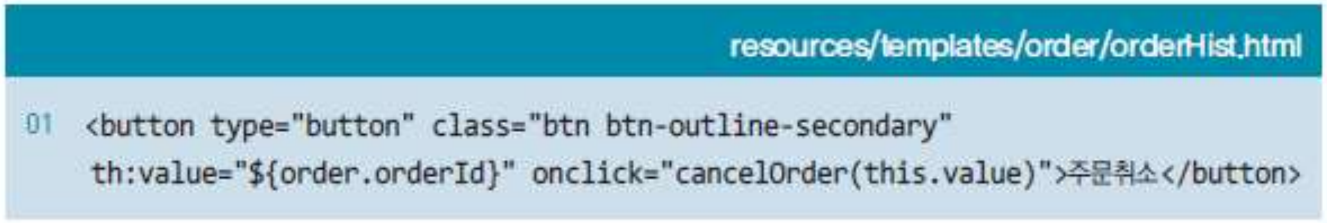
```
<!-- 사용자 스크립트 추가 -->
<th:block layout:fragment="script">

    <script th:inline="javascript">
        function cancelOrder(orderId) {
            var token = $("meta[name='_csrf']").attr("content");
            var header = $("meta[name='_csrf_header']").attr("content");

            var url = "/order/" + orderId + "/cancel";
            var paramData = {                                          ❶
                orderId : orderId,
            };

            var param = JSON.stringify(paramData);

```

7.3 주문 취소

```
            $.ajax({
                url      : url,
                type     : "POST",
                contentType : "application/json",
                data     : param,
                beforeSend : function(xhr){
                    /* 데이터를 전송하기 전에 헤더에 csrf 값을 설정 */
                    xhr.setRequestHeader(header, token);
                },
                dataType : "json",
                cache   : false,
                success  : function(result, status){ ❷
                    alert("주문이 취소 되었습니다.");
                    location.href='/orders/' + [[${page}]];
                },
                error : function(jqXHR, status, error){
                    if(jqXHR.status == '401'){
                        alert('로그인 후 이용해주세요');
                        location.href='/members/login';
                    } else{
                        alert(jqXHR.responseText);
                    }
                }
            });
        }
    </script>

</th:block>
```

7.3 주문 취소

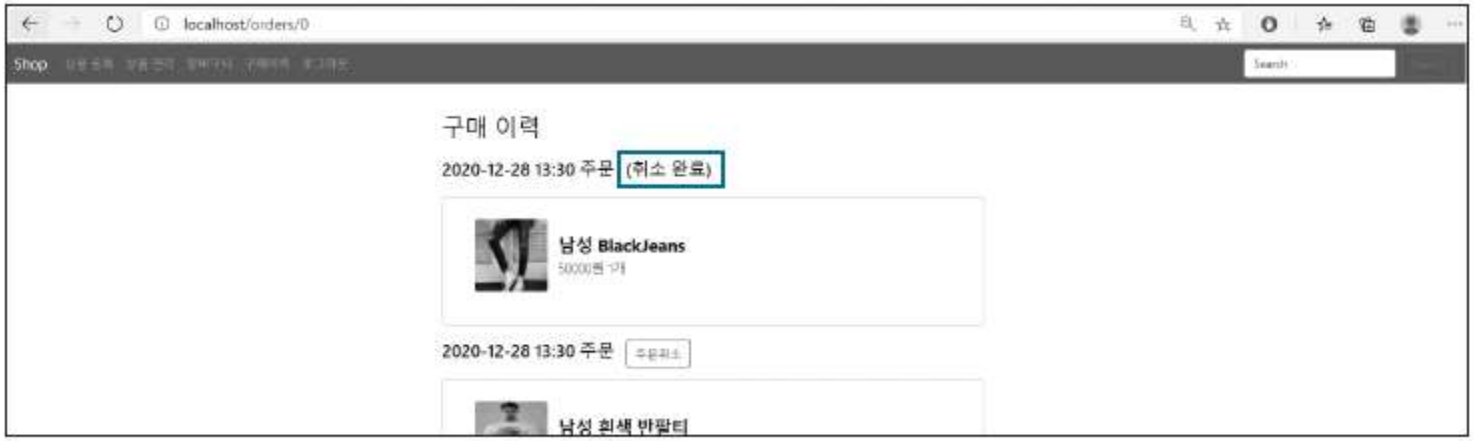
- 주문 취소 버튼 클릭 시 cancelOrder함수 호출



resources/templates/order/orderHist.html

```
<button type="button" class="btn btn-outline-secondary"
 th:value="${order.orderId}" onclick="cancelOrder(this.value)">주문취소</button>
```

7.3 주문 취소



구매 이력

2020-12-28 13:30 주문 (취소 완료)

남성 BlackJeans

50000원 1개

2020-12-28 13:30 주문 주문취소

남성 흰색 반팔티

[그림 7-8] 주문 취소 완료

Thank you for your attention