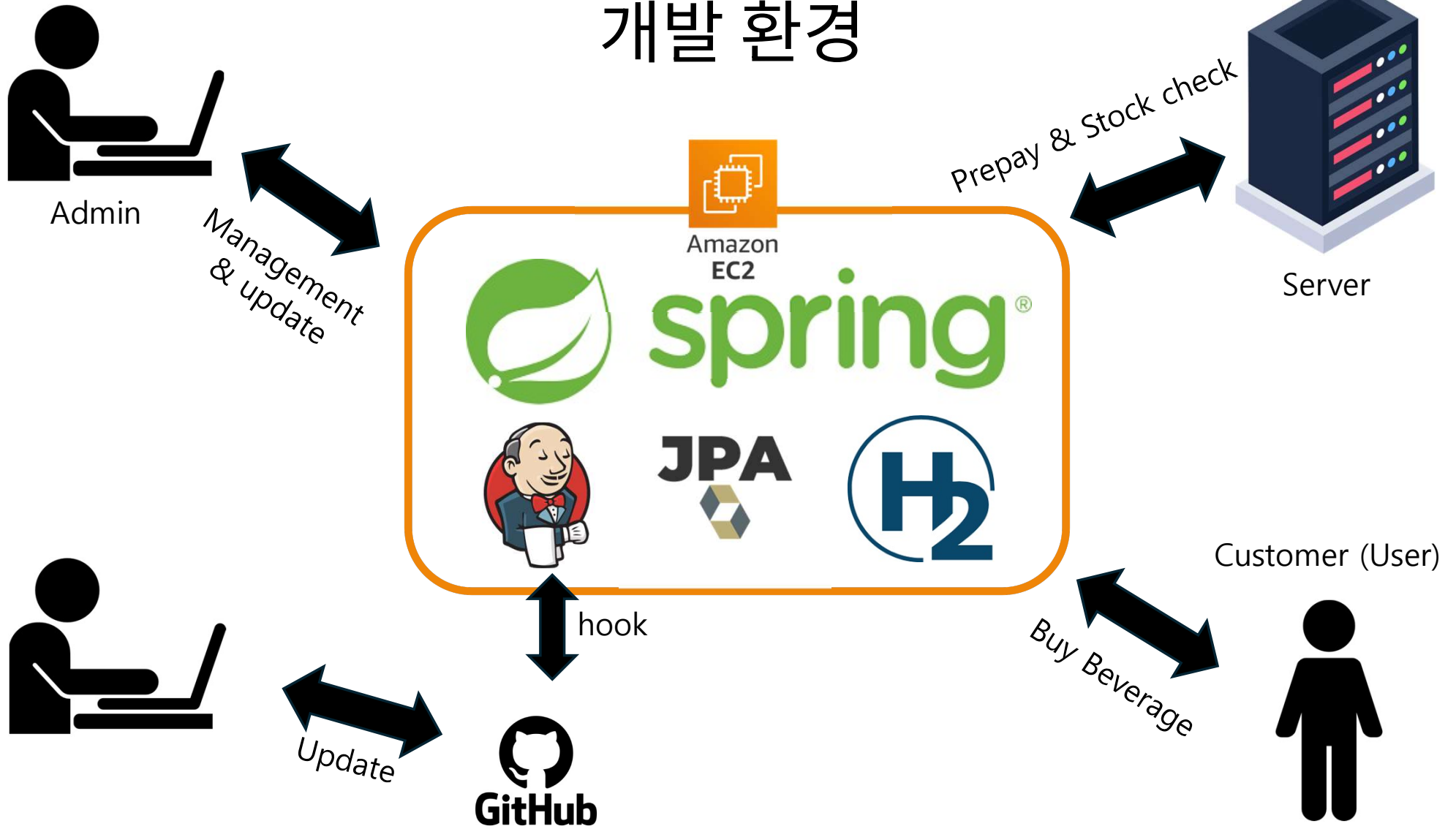


# Team 5 2050 Presentation

-202011256 김대원  
-202011309 성승재  
-202115009 성정현  
-202011343 이은기

# 개발 환경



# 세부 CI/CD (Jenkins)

Web hook으로 push 될 때 마다 알림



## Webhooks

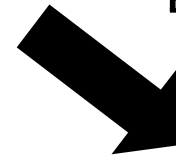
Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

• <http://13.124.215.64:8080/github-we...> (push)

Edit Delete

알림을 받고 update



Dashboard > DVM > Configuration

### Configure

None  
 Git ?

Repositories ?

Repository URL ?

Credentials ?

+ Add ▾  
고급 ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

Add Branch

Repository browser ?

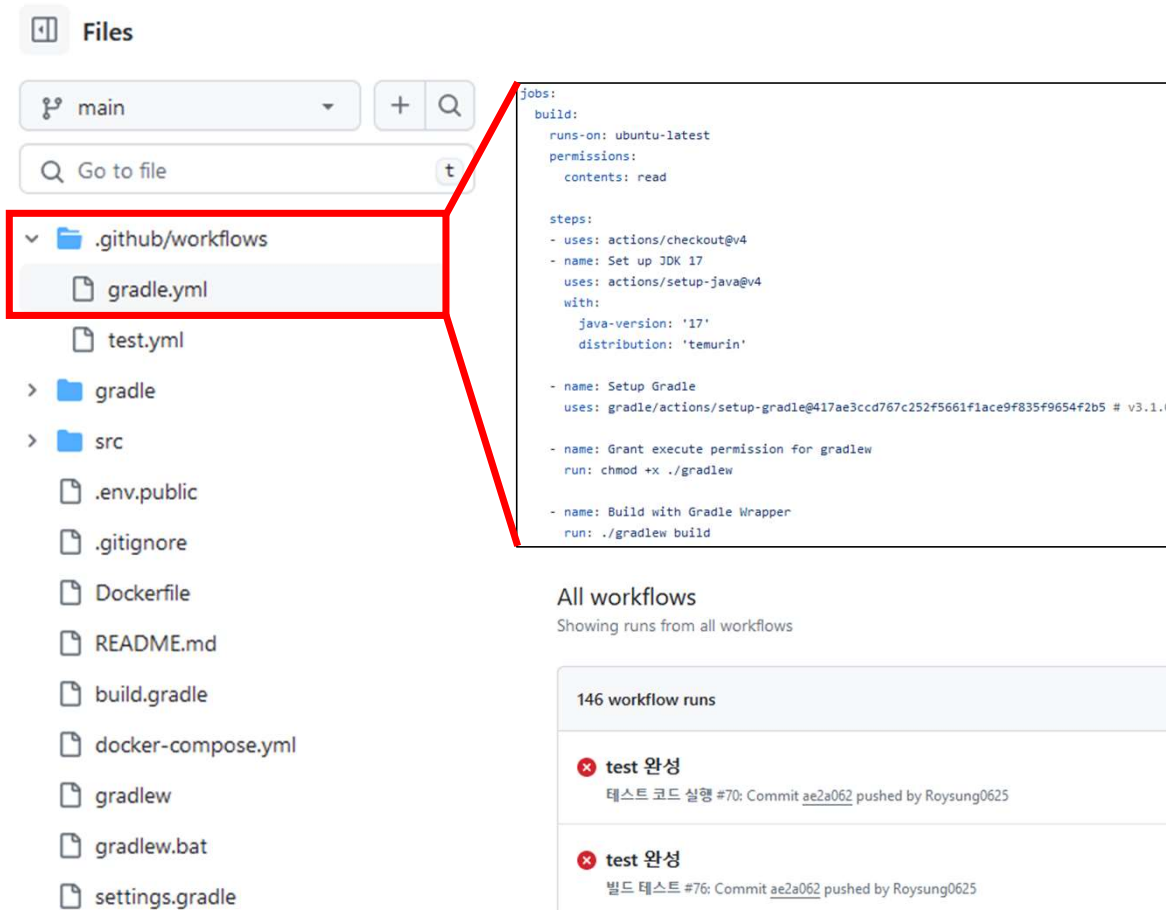
### Build History

추이 ▾

Filter...

✓ #28	<a href="#">2024. 6. 9. 오후 1:56</a>
✓ #27	<a href="#">2024. 6. 9. 오후 1:56</a>
✓ #26	<a href="#">2024. 6. 9. 오후 1:56</a>
✓ #25	<a href="#">2024. 6. 9. 오후 1:56</a>
✓ #24	<a href="#">2024. 6. 9. 오후 1:56</a>
✓ #23	<a href="#">2024. 6. 9. 오후 1:55</a>

# 배포 Unit Test



```
jobs:  
  build:  
    runs-on: ubuntu-latest  
    permissions:  
      contents: read  
  
    steps:  
      - uses: actions/checkout@v4  
      - name: Set up JDK 17  
        uses: actions/setup-java@v4  
        with:  
          java-version: '17'  
          distribution: 'temurin'  
  
      - name: Setup Gradle  
        uses: gradle/actions/setup-gradle@417ae3ccd767c252f5661f1ace9f835f9654f2b5 # v3.1.0  
  
      - name: Grant execute permission for gradlew  
        run: chmod +x ./gradlew  
  
      - name: Build with Gradle Wrapper  
        run: ./gradlew build
```



All workflows  
Showing runs from all workflows

146 workflow runs

Event	Status	Branch	Actor
test 완성	Success	main	...
테스트 코드 실행 #70: Commit <a href="#">ae2a062</a> pushed by Roysung0625			
test 완성	Success	main	...
빌드 테스트 #76: Commit <a href="#">ae2a062</a> pushed by Roysung0625			



# Unit Test

Test Results	2 sec 265 ms
SocketServerStockTest	439 ms
소켓 통신 테스트 - dst_id가 다른 경우 req_stock 메시지 전송	429 ms
소켓 통신 테스트 - req_stock 메시지 전송	6 ms
소켓 통신 테스트 - Hello 메시지 전송	4 ms
DemoApplicationTests	2 ms
contextLoads	2 ms
ManagementControllerTest	499 ms
getItemsByDVM	398 ms
getAllCodes	8 ms
getAllItems	22 ms
updateItem	63 ms
getItemByItemCode	8 ms
MenuControllerTest	19 ms
menu 진입	9 ms
Admin monitoring 진입	4 ms
management 진입	4 ms
code 입력 진입	2 ms
MonitorControllerTest	30 ms
getAllDVM	25 ms
getMyInfo	5 ms
PaymentControllerTest	60 ms
pickup	26 ms
prepay	26 ms
pay	8 ms

```
@DisplayName("소켓 통신 테스트 - req_stock 메시지 전송")
@Test
public void requestStock() throws Exception {
    initClient();
    // 메시지 구성
    HashMap<String, String> stock = new HashMap<>();
    stock.put("item_code", "1");
    stock.put("item_num", "3");
    // 메시지 전송 - team1에서 team5로 요청
    SocketMessage message = new SocketMessage(msgType: "req_stock", srcId: "team1", dstId: "team5", stock);
    output.println(message.toJson());
    // 응답 확인
    SocketMessage resp = SocketMessage.fromJson(input.readLine());
    assert Objects.equals(resp.getMsg_type(), b: "resp_stock") : "응답: " + resp.toJson();
    assert Objects.equals(resp.getSrc_id(), b: "team5") : "응답: " + resp.toJson();
    assert Objects.equals(resp.getDst_id(), b: "team1") : "응답: " + resp.toJson();
    assert Objects.equals(resp.getMsg_content().get("item_code"), b: "01") : "응답: " + resp.toJson();
    assert Objects.equals(resp.getMsg_content().get("item_num"), b: "10") : "응답: " + resp.toJson();
}
```

Local Socket을 이용한 Test

MockMvc를 이용한 Controller 응답 Test

```
@Test
void getAllItems() {
    try {
        mvc.perform(MockMvcRequestBuilders.get(uriTemplate: "/items"))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression: "$").exists());
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

# Unit Test

✓ CodeTest	0ms
✓ Code 생성 테스트 - 성공 케이스	0ms
✓ MyInfoTest	2ms
✓ Test getInfo	2ms
✓ SocketMessageTest	5ms
✓ 단순 메시지 파싱 테스트 - 성공 케이스	2ms
✓ 재고확인 요청 메시지	0ms
✓ 재고확인 응답 메시지	0ms
✓ 선결제 요청 메시지	1ms
✓ 선결제 가능 여부 응답 메시지	1ms
✓ 단순 메시지 생성 테스트 - 성공 케이스	1ms
✓ CodeRepositoryTest	43ms
✓ code 저장 테스트	32ms
✓ 시간 이전 code 조회 테스트	11ms
✓ ItemRepositoryTests	64ms
✓ item 저장 테스트	21ms
✓ ItemCode로 조회 테스트	8ms
✓ ItemCode로 조회시 오류 테스트	7ms
✓ item 개수 수정 테스트	28ms
✓ CodeServiceTest	65ms
✓ 시간이 지난 코드 삭제 테스트	48ms
✓ 시간이 지난 코드 삭제되고 item 복구 테스트	17ms
✓ PaymentServiceTest	16ms
✓ pick up test	16ms
✓ SocketServerPrepayTest	10ms
✓ 소켓 통신 테스트 - req_prepay 메시지 전송	10ms
✓ SocketServerTest	1 sec 11 ms
✓ 별도 생성된 소켓 서버 테스트 - 조회	1 sec 11 ms

```
public class CodeTest {
    @GreenRain
    @DisplayName("Code 생성 테스트 - 성공 케이스")
    @Test
    public void testCreateCode() {
        LocalDateTime time = LocalDateTime.now();
        Code code = new Code(code: "code1", time, itemCode: 1, quantity: 1);
        assert code.getCode().equals("code1");
        assert code.getTime().equals(time);
        assert code.getItemCode() == 1;
        assert code.getQuantity() == 1;
    }
}
```

Domain이 제대로 작동하는지 Test

Gson을 이용하여 Class가 제대로 Json으로 되는지 Test

```
@DisplayName("단순 메시지 생성 테스트 - 성공 케이스")
@Test
public void testToJson() {
    @GreenRain
    SocketMessage message = new SocketMessage(msgType: "type", srcId: "src", dstId: "dst", new HashMap<>() {
        {
            put("msg_content", "content");
        }
    });
    String json = message.toJson();
    assert json.equals("{\"msg_type\":\"type\",\"src_id\":\"src\",\"dst_id\":\"dst\",\"msg_content\":{\"content\":\"content\"}}");
}
```

# Unit Test

✓ CodeTest	0 ms
✓ Code 생성 테스트 - 성공 케이스	0 ms
✓ MyInfoTest	2 ms
✓ Test getInfo	2 ms
✓ SocketMessageTest	5 ms
✓ 단순 메시지 파싱 테스트 - 성공 케이스	2 ms
✓ 재고확인 요청 메시지	0 ms
✓ 재고확인 응답 메시지	0 ms
✓ 선결제 요청 메시지	1 ms
✓ 선결제 가능 여부 응답 메시지	1 ms
✓ 단순 메시지 생성 테스트 - 성공 케이스	1 ms
✓ CodeRepositoryTest	43 ms
✓ code 저장 테스트	32 ms
✓ 시간 이전 code 조회 테스트	11 ms
✓ ItemRepositoryTests	64 ms
✓ item 저장 테스트	21 ms
✓ ItemCode로 조회 테스트	8 ms
✓ ItemCode로 조회시 오류 테스트	7 ms
✓ item 개수 수정 테스트	28 ms
✓ CodeServiceTest	65 ms
✓ 기간이 지난 코드 삭제 테스트	48 ms
✓ 기간이 지난 코드 삭제되고 item 복구 테스트	17 ms
✓ PaymentServiceTest	16 ms
✓ pick up test	16 ms
✓ SocketServerPrepayTest	10 ms
✓ 소켓 통신 테스트 - req_prepay 메시지 전송	10 ms
✓ SocketServerTest	1 sec 11 ms
✓ 별도 생성된 소켓 서버 테스트 - 조회	1 sec 11 ms

```
@Test
public void saveCode() {
    LocalDateTime time = LocalDateTime.now();
    // 초기 값이 있는지 확인
    assert codeRepository.findAll().isEmpty() : "실제 값 : " + codeRepository.findAll();
    // 저장
    Code testCode = new Code(code: "testCode1", time, itemCode: 1, quantity: 1);
    codeRepository.save(testCode);
    Code savedCode = codeRepository.findByCode("testCode1");

    // 저장된 데이터가 맞는지 확인
    assert savedCode.getCode().equals("testCode1") : "실제 값 : " + savedCode.getCode();
    assert savedCode.getTime().equals(time) : "실제 값 : " + savedCode.getTime();
    assert savedCode.getItemCode() == 1 : "실제 값 : " + savedCode.getItemCode();
    assert savedCode.getQuantity() == 1 : "실제 값 : " + savedCode.getQuantity();
}
```

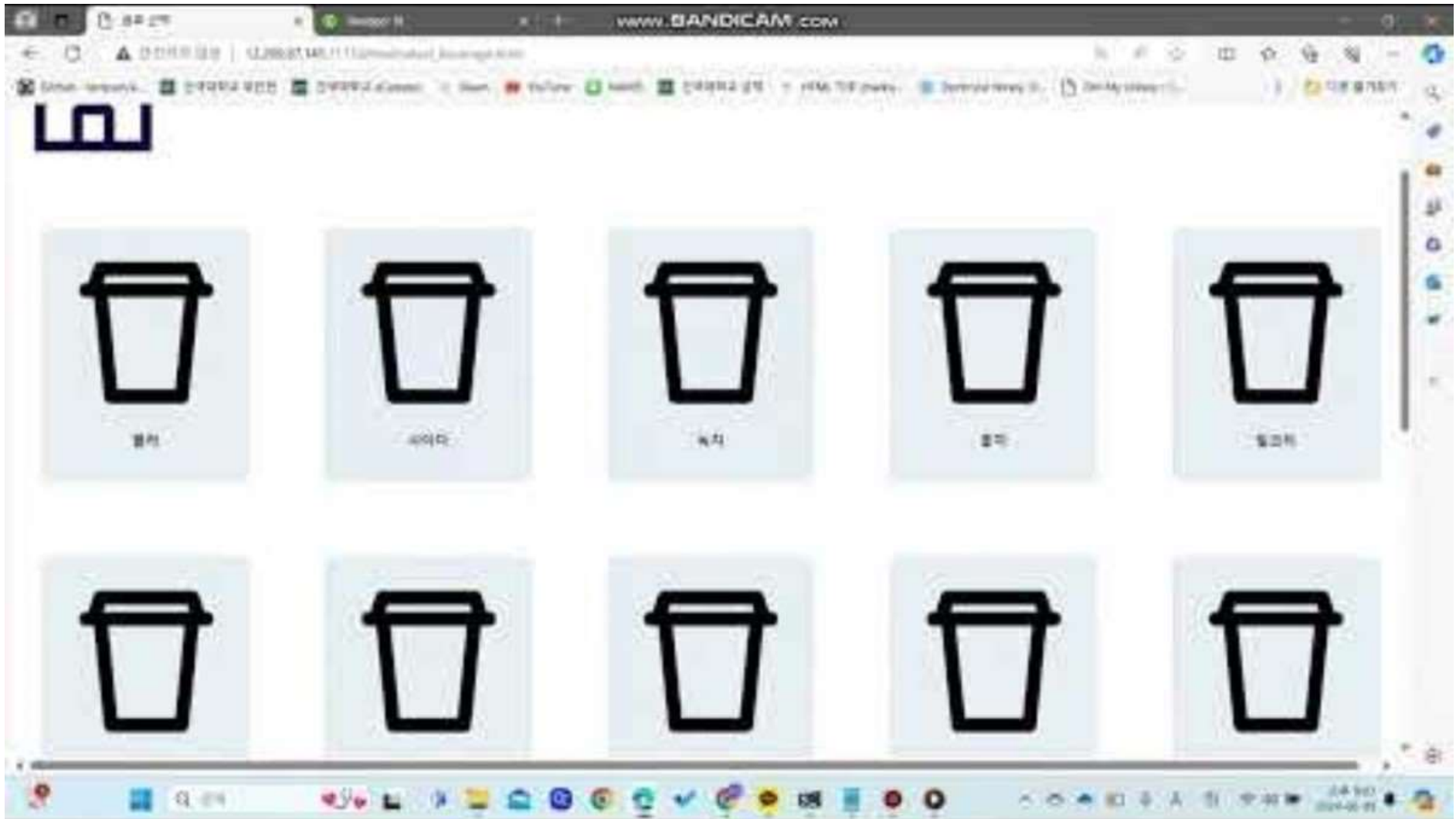
JPA를 활용한 저장이 정상적으로 진행되는지 Test

비즈니스 로직이 정상적으로 동작하는지 Test

```
@DisplayName("기간이 지난 코드 삭제 테스트")
@Test
public void deleteExpiredCodesTest() {
    // given
    LocalDateTime oldDate = LocalDateTime.now().minusDays(31);
    codeRepository.save(new Code(code: "code1", oldDate, itemCode: 9, quantity: 1));
    codeRepository.save(new Code(code: "code2", LocalDateTime.now(), itemCode: 9, quantity: 2));
    // when
    managementService.deleteExpiredCodes();
    // then
    Code code1 = codeRepository.findByCode("code1");
    Code code2 = codeRepository.findByCode("code2");
    assert code1 == null : "기간이 지난 코드가 삭제되지 않았습니다." + code1;
    assert code2 != null : "기간이 지나지 않은 코드가 삭제되었습니다." + null;
}
```

# Demo 영상

<https://youtu.be/GlQ49-KtyZo?si=SjFHuZQ-WxgaAy8B>

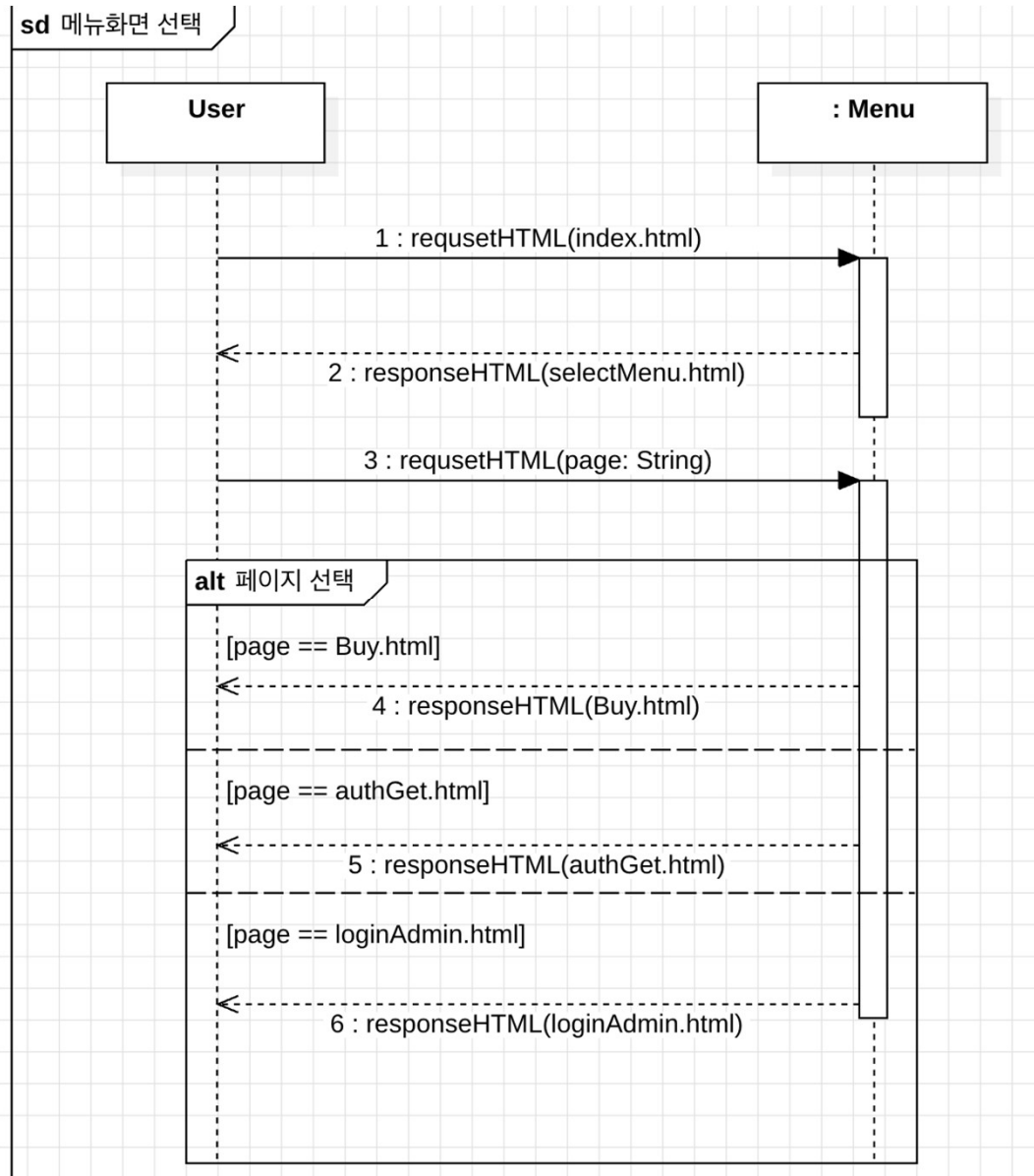


# Interaction Diagrams

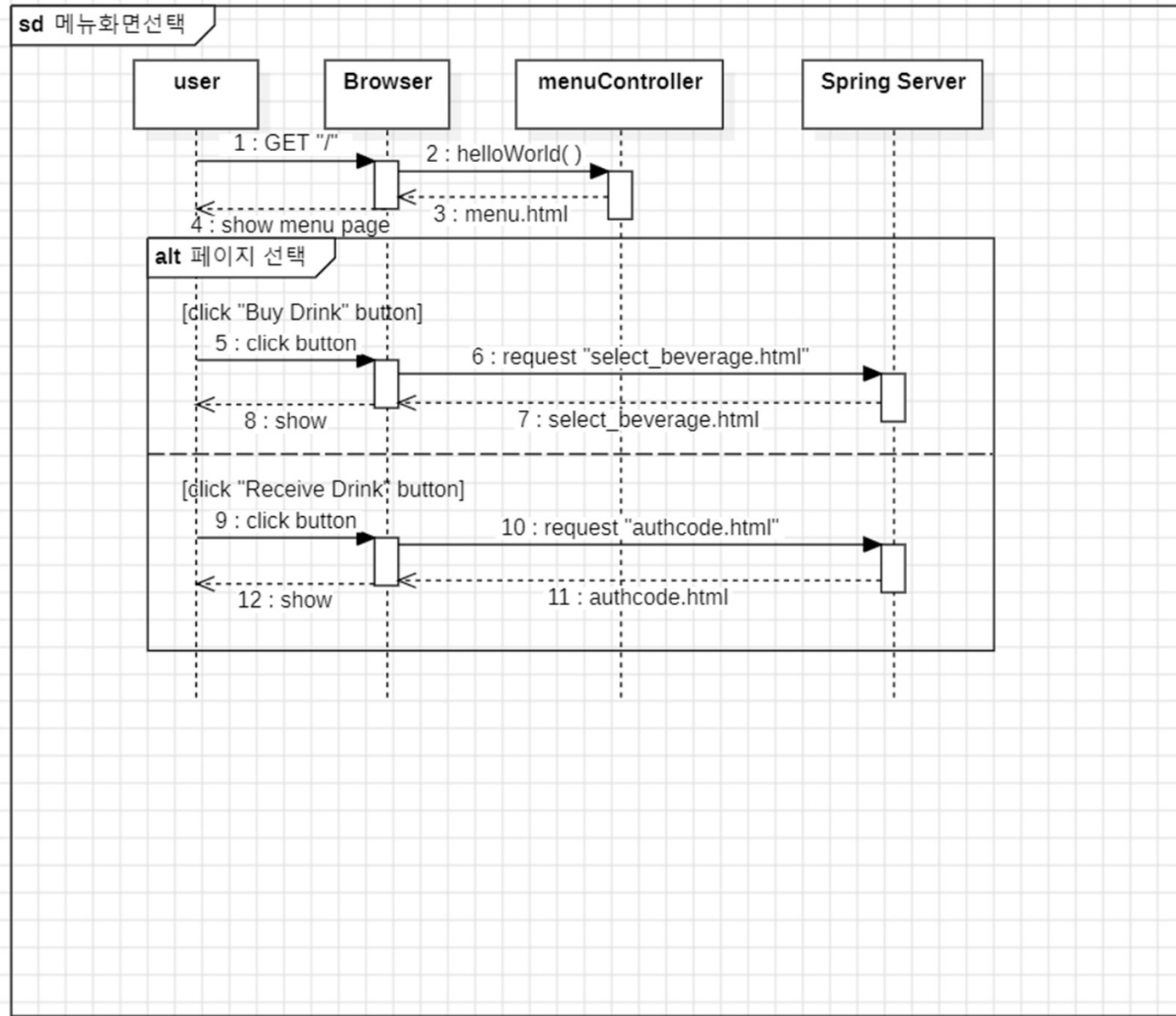
에서 변경된 부분



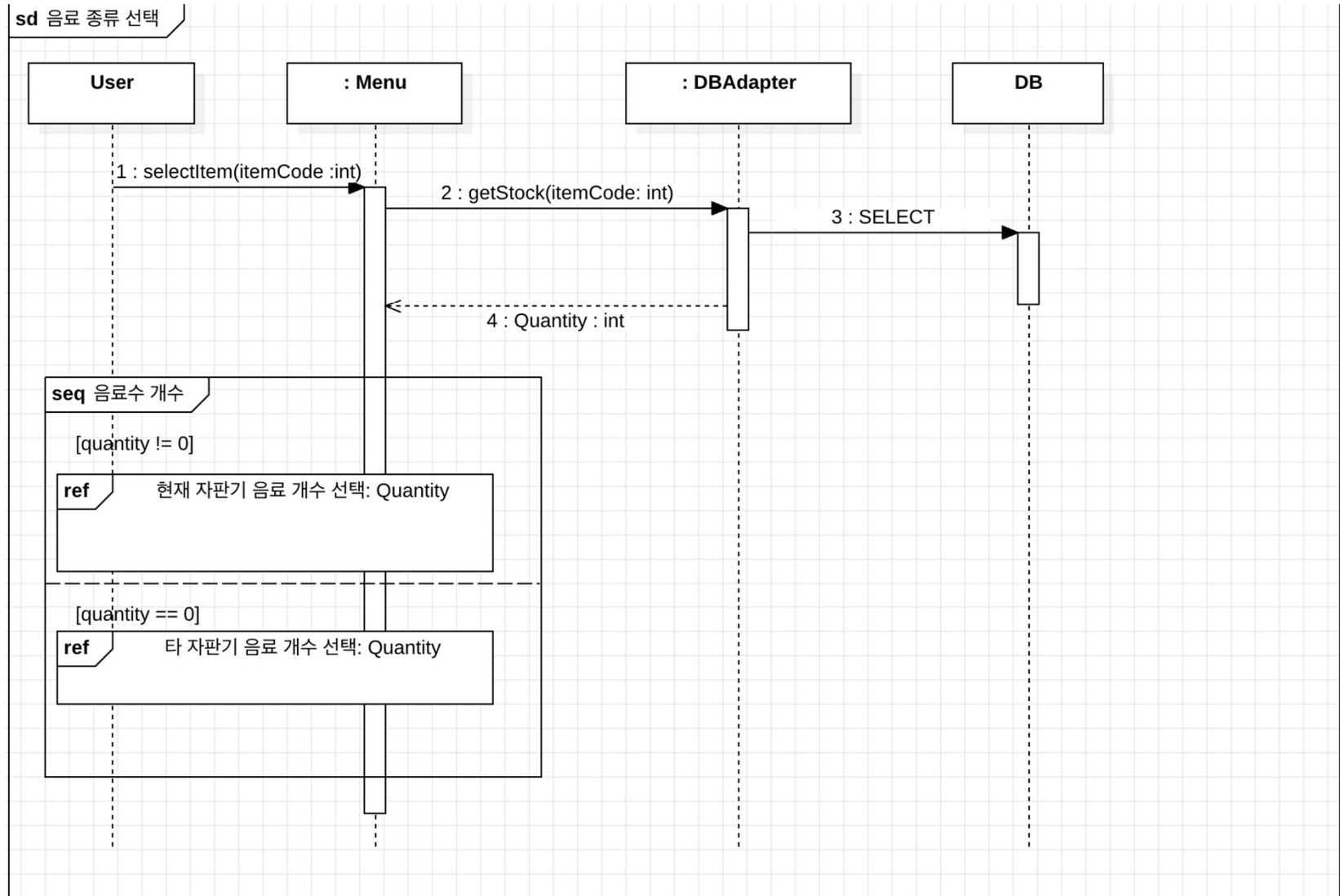
# 변경 전



# 변경 후

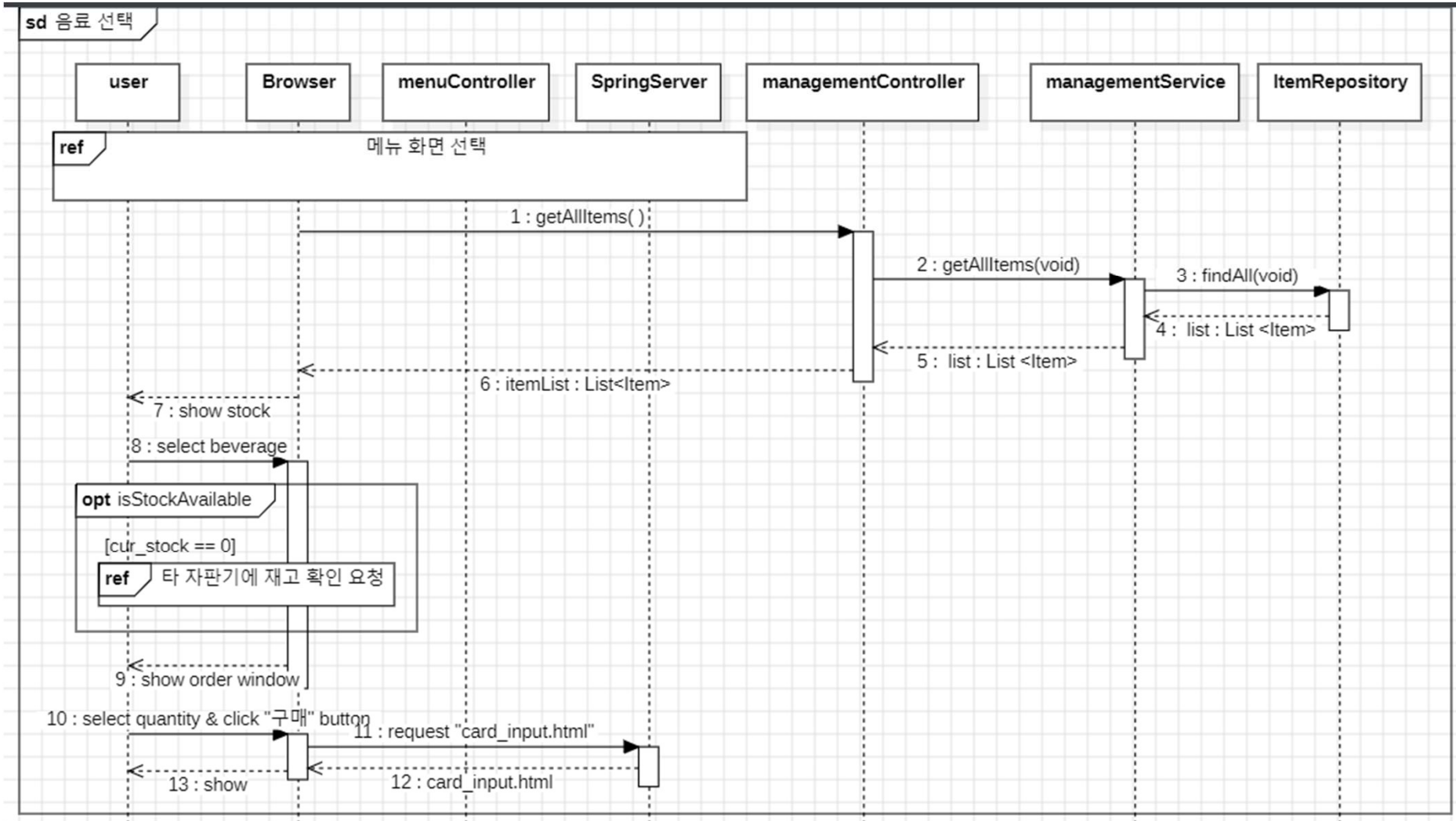


# 변경 전



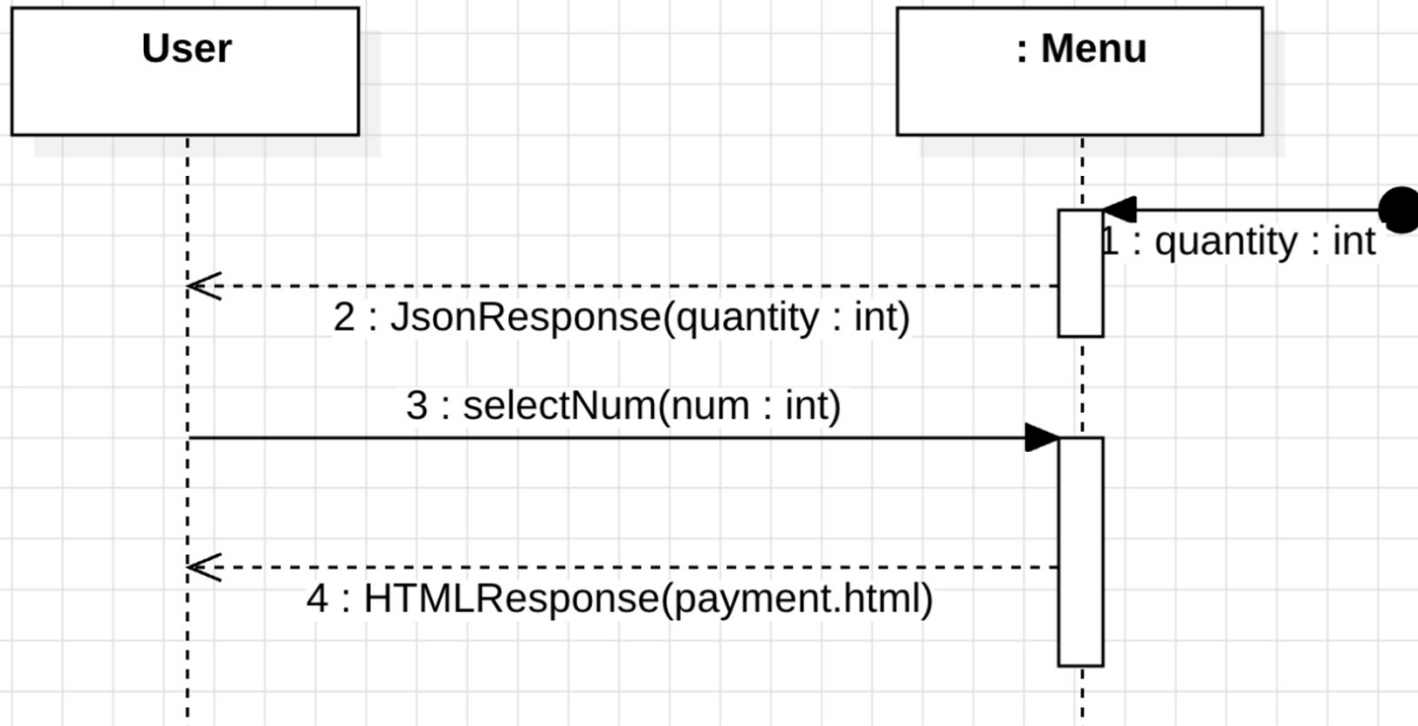


# 변경 후

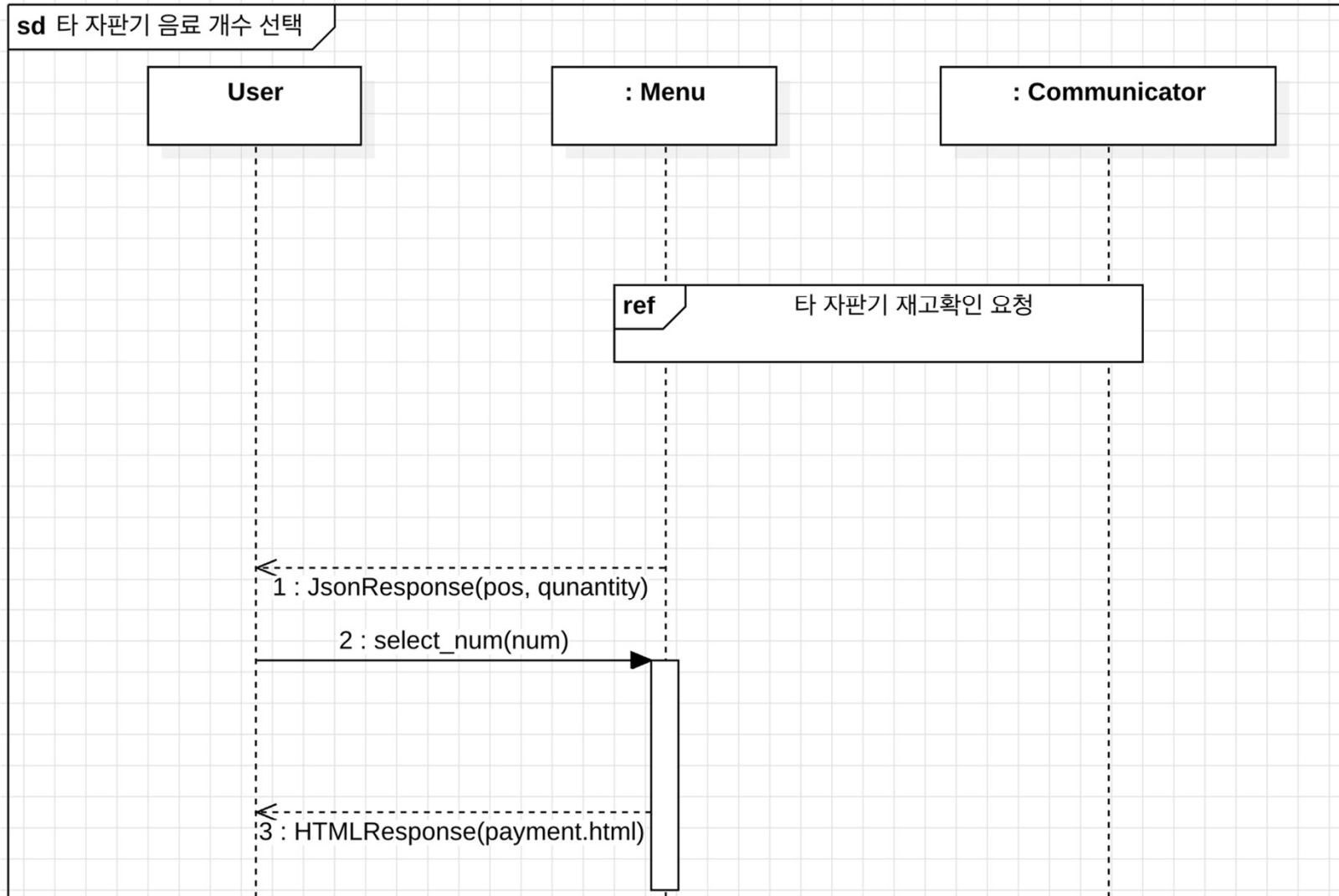


# 변경 후 삭제

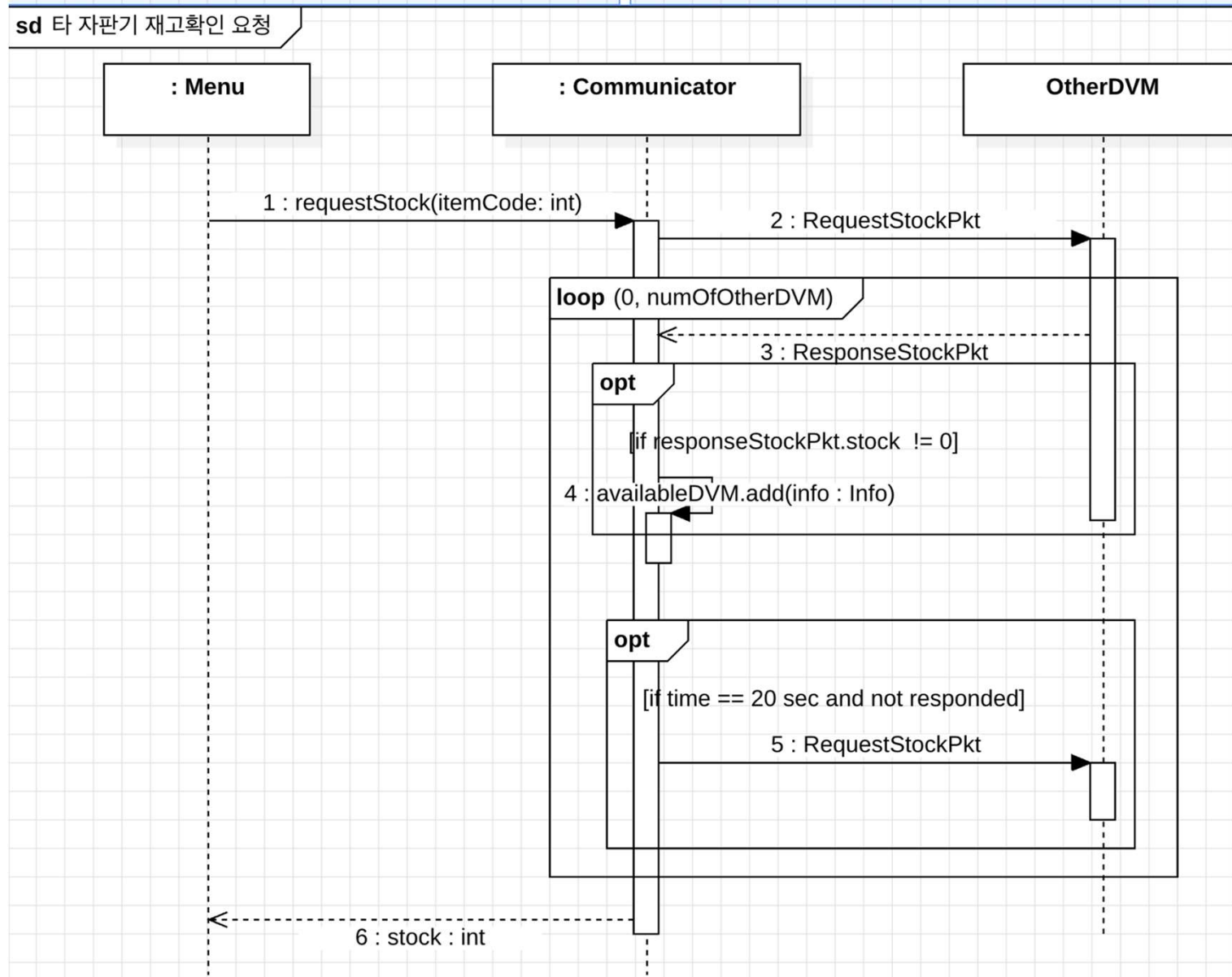
sd 현재 자판기 음료 개수 선택



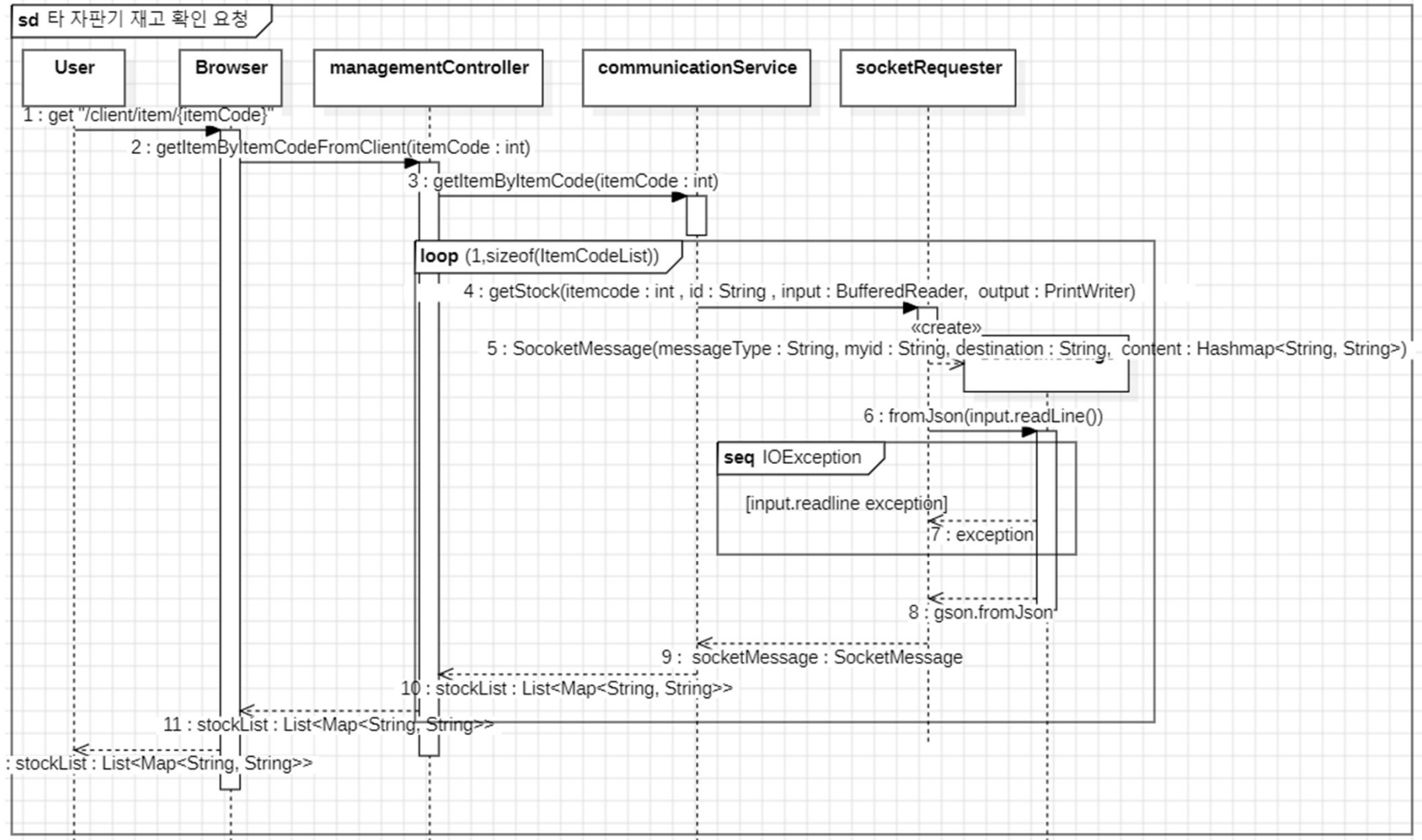
# 변경 후 삭제



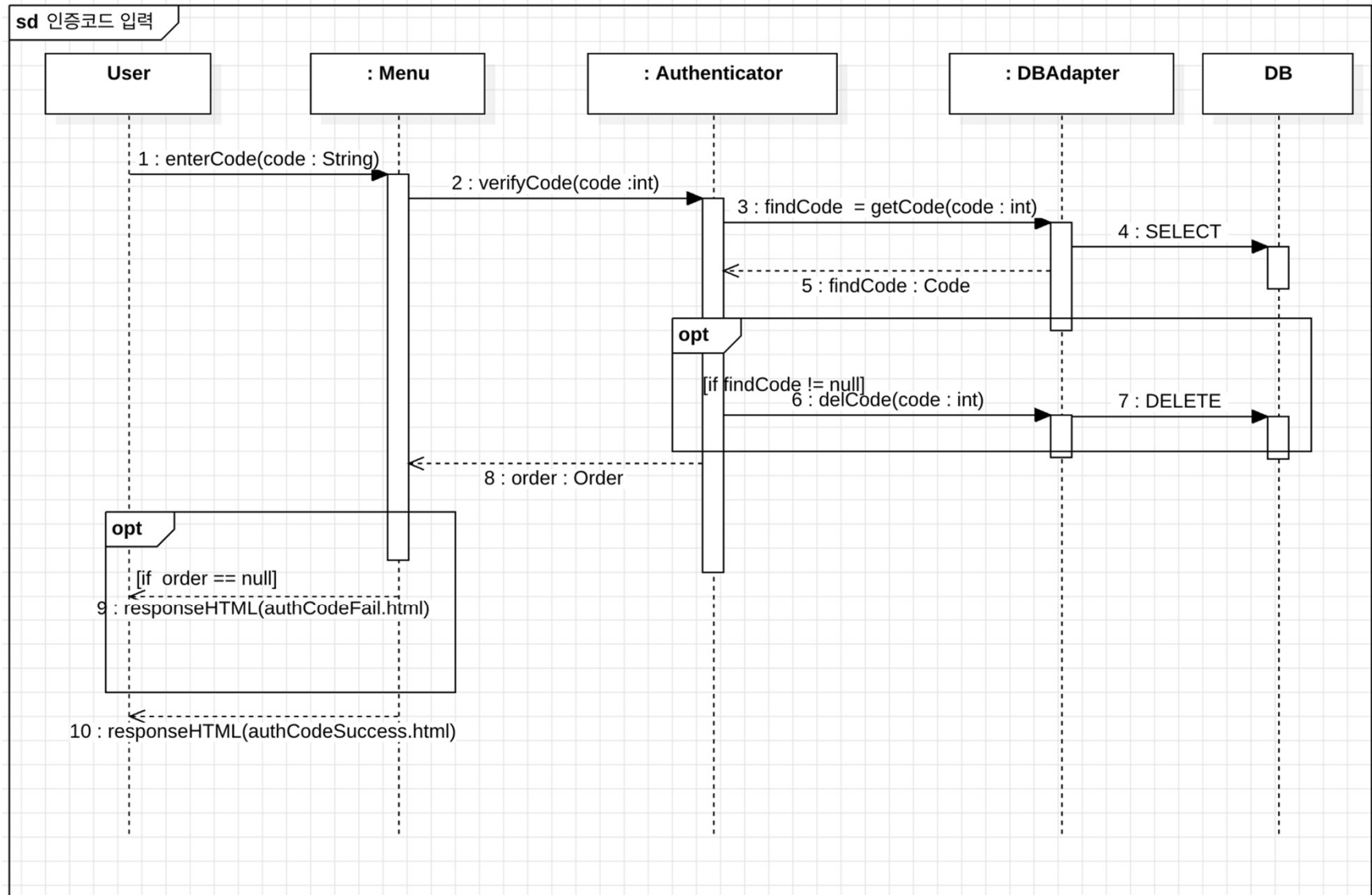
# 변경 전



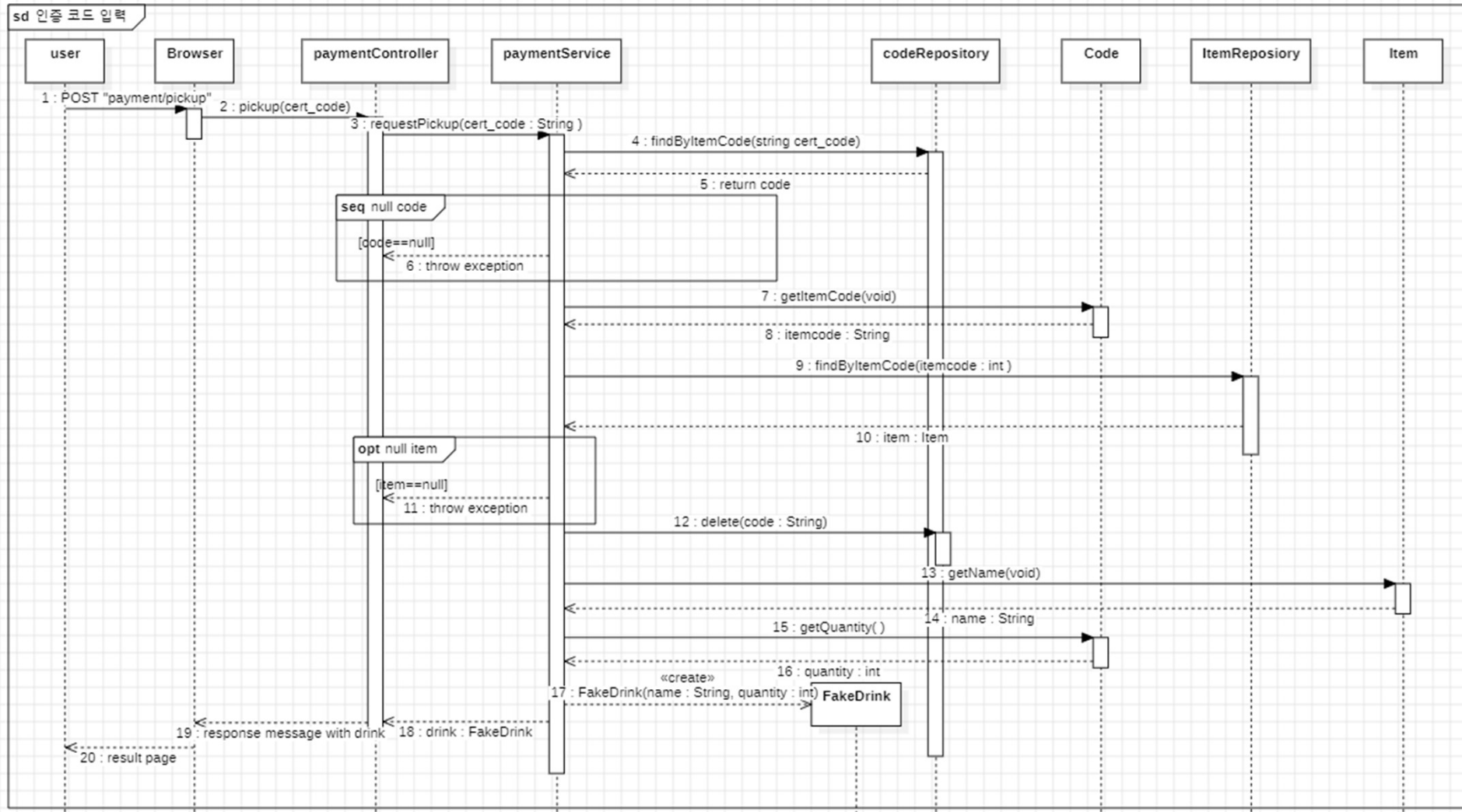
# 변경 후



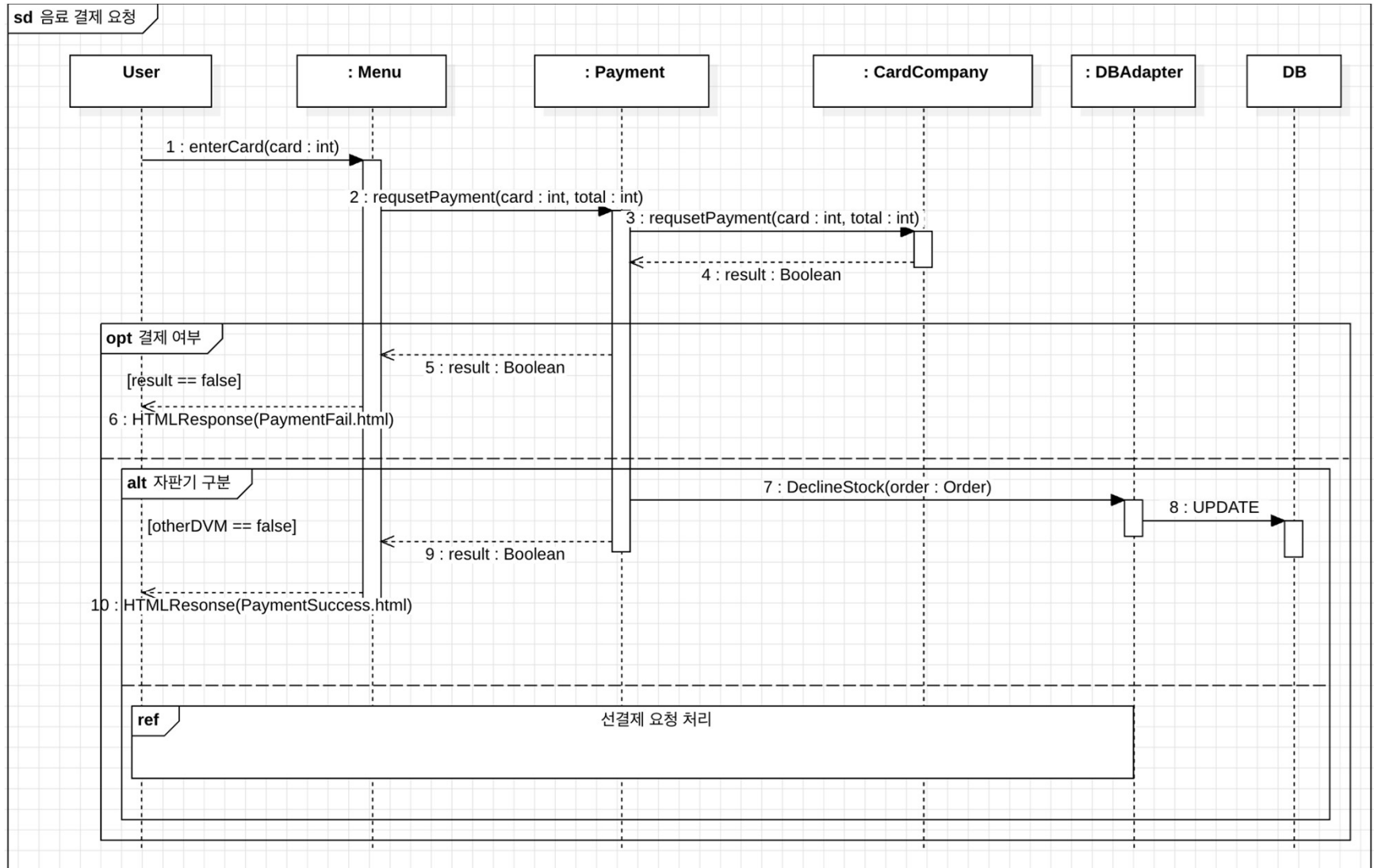
# 변경 전



# 변경 후

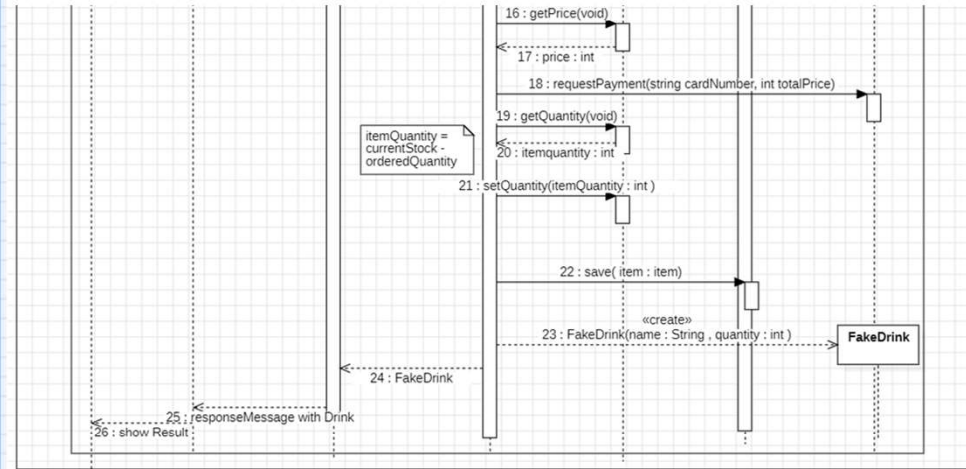
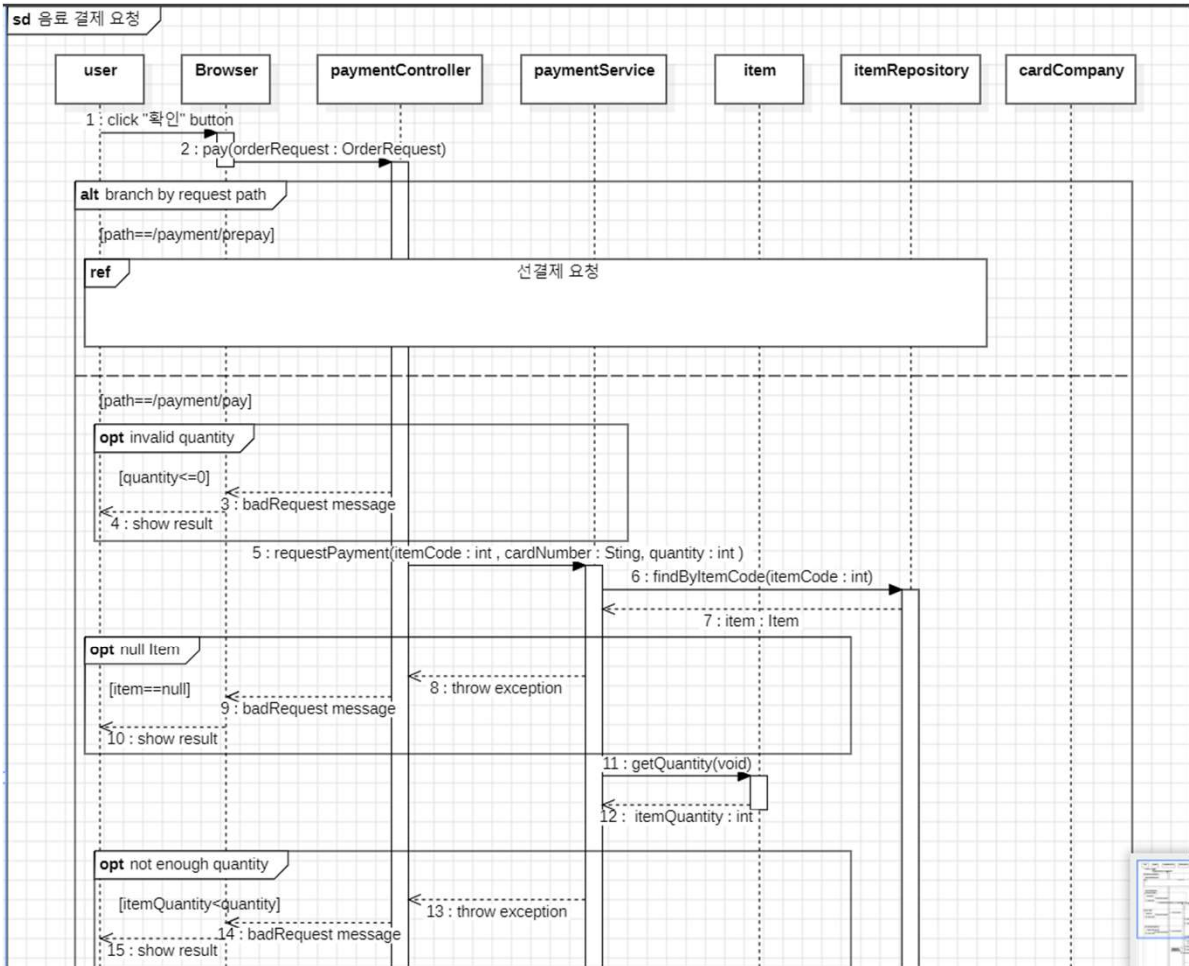


# 변경 전

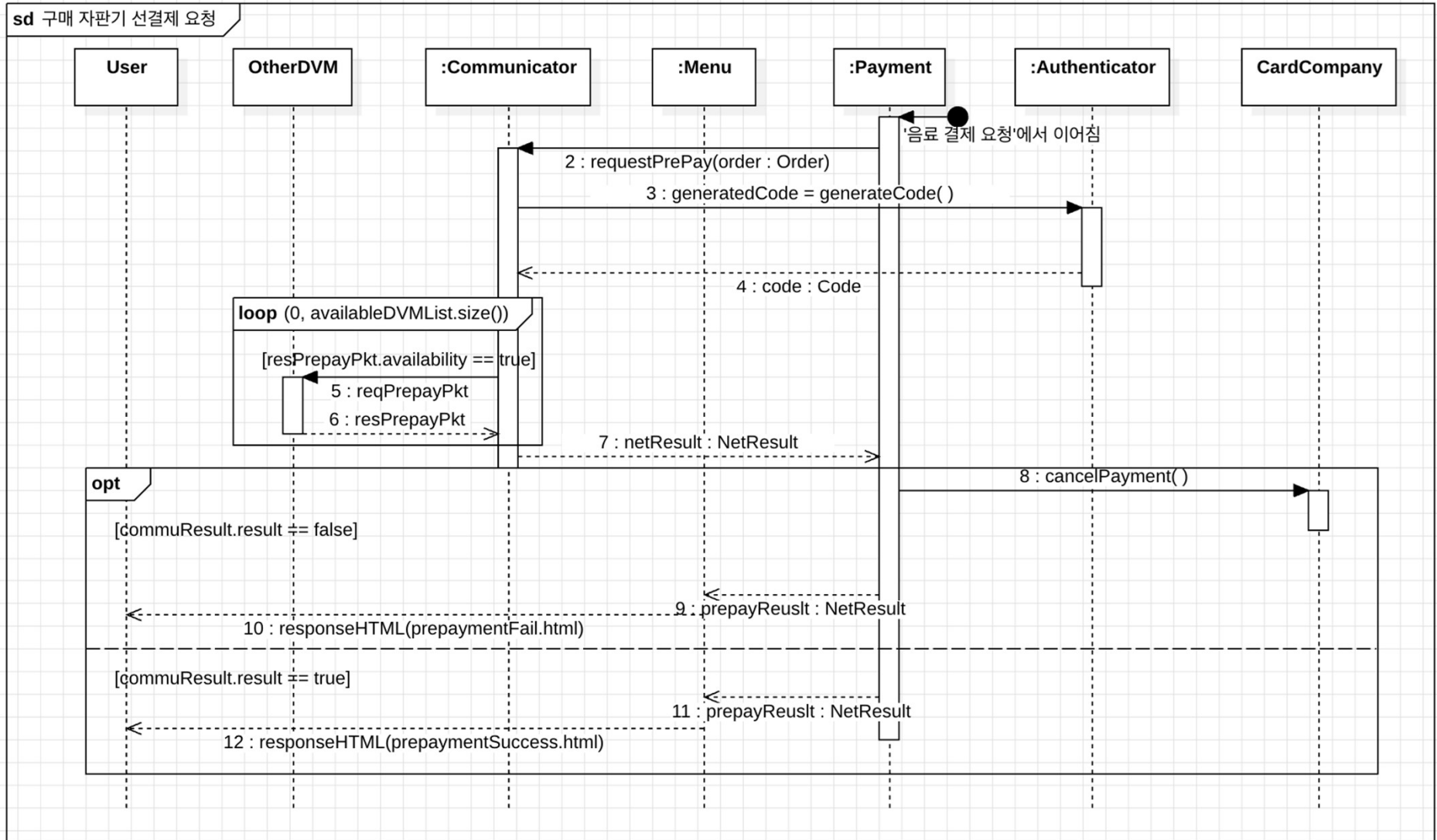




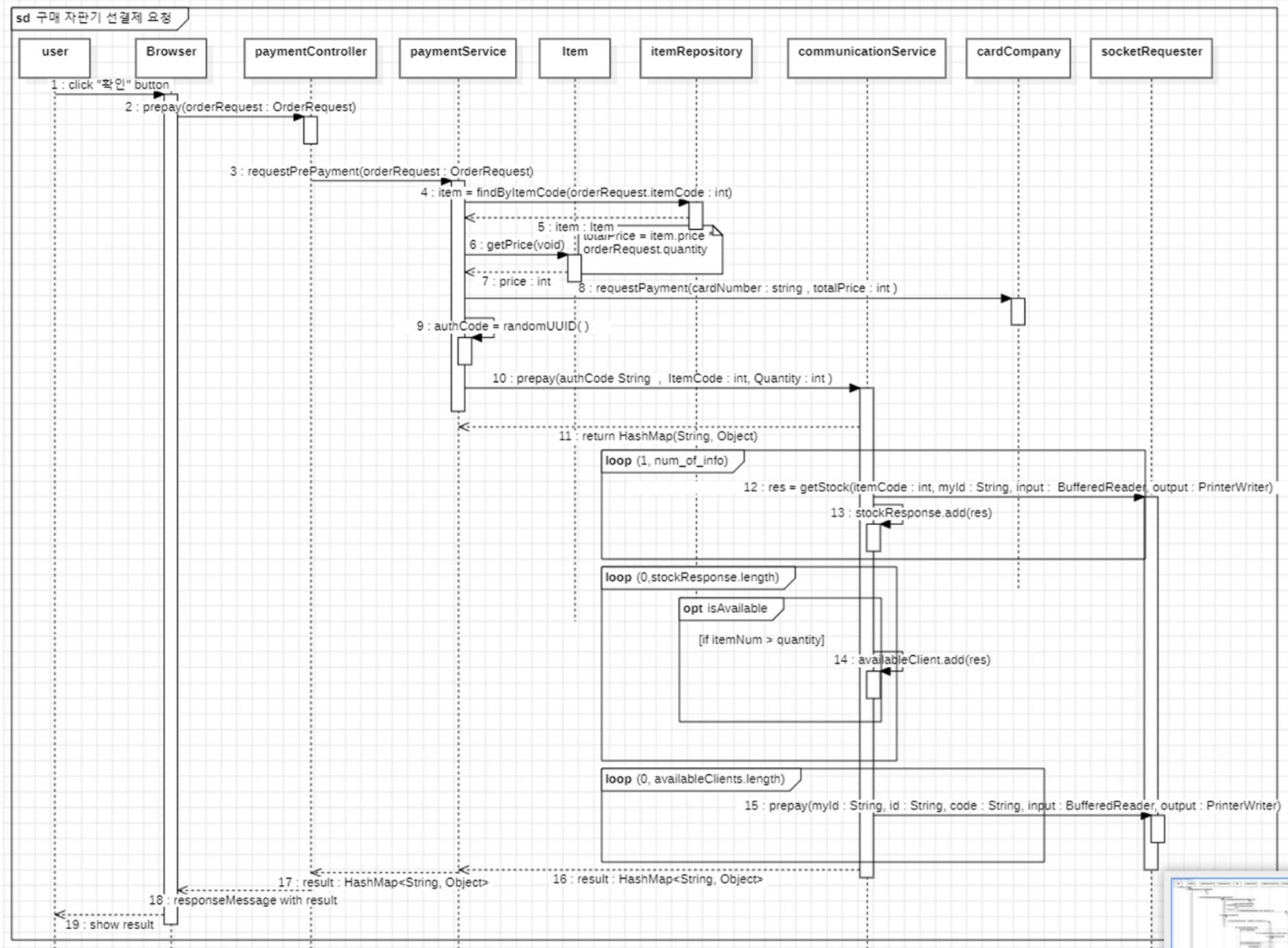
# 변경 후



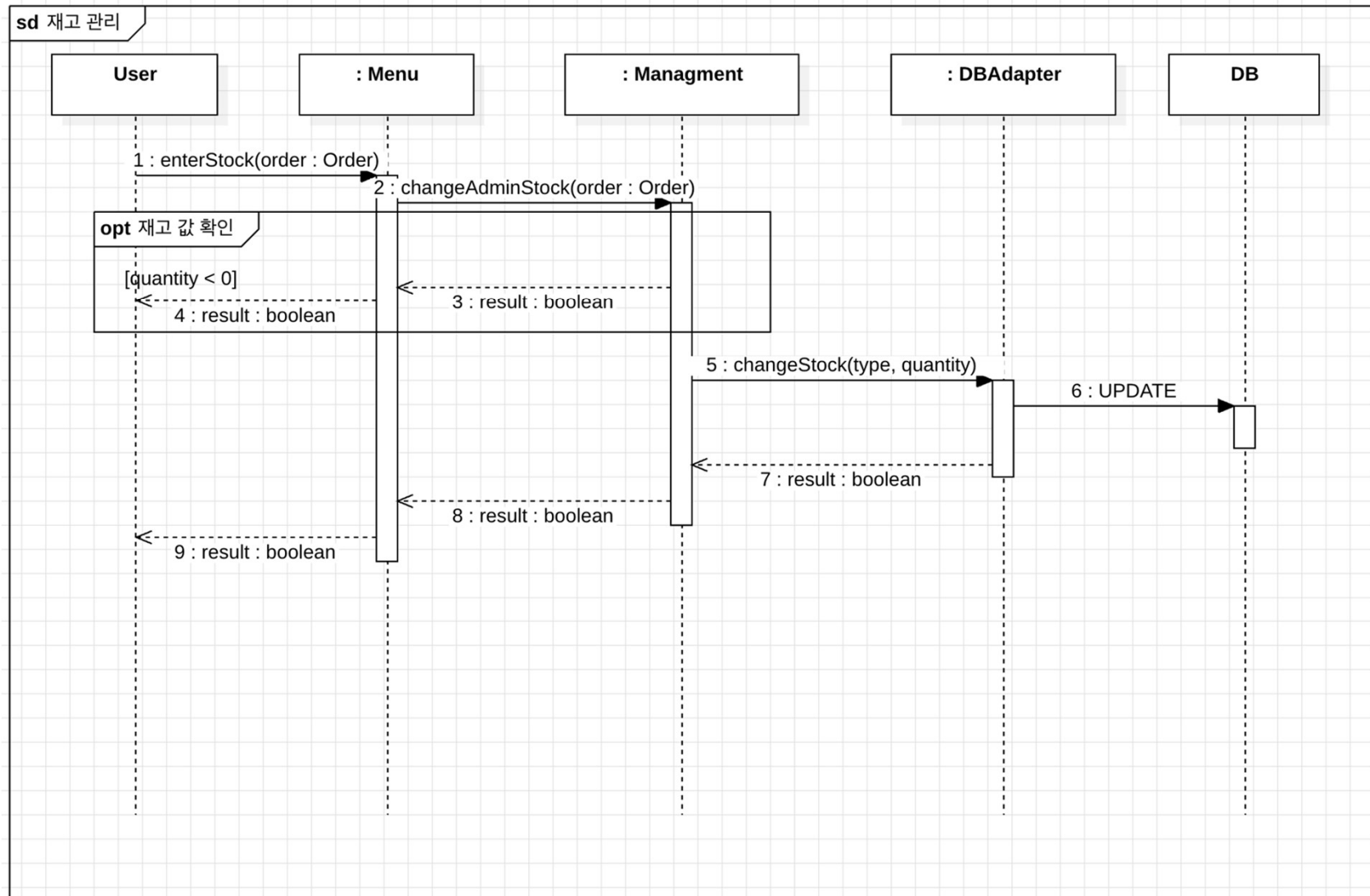
# 변경 전



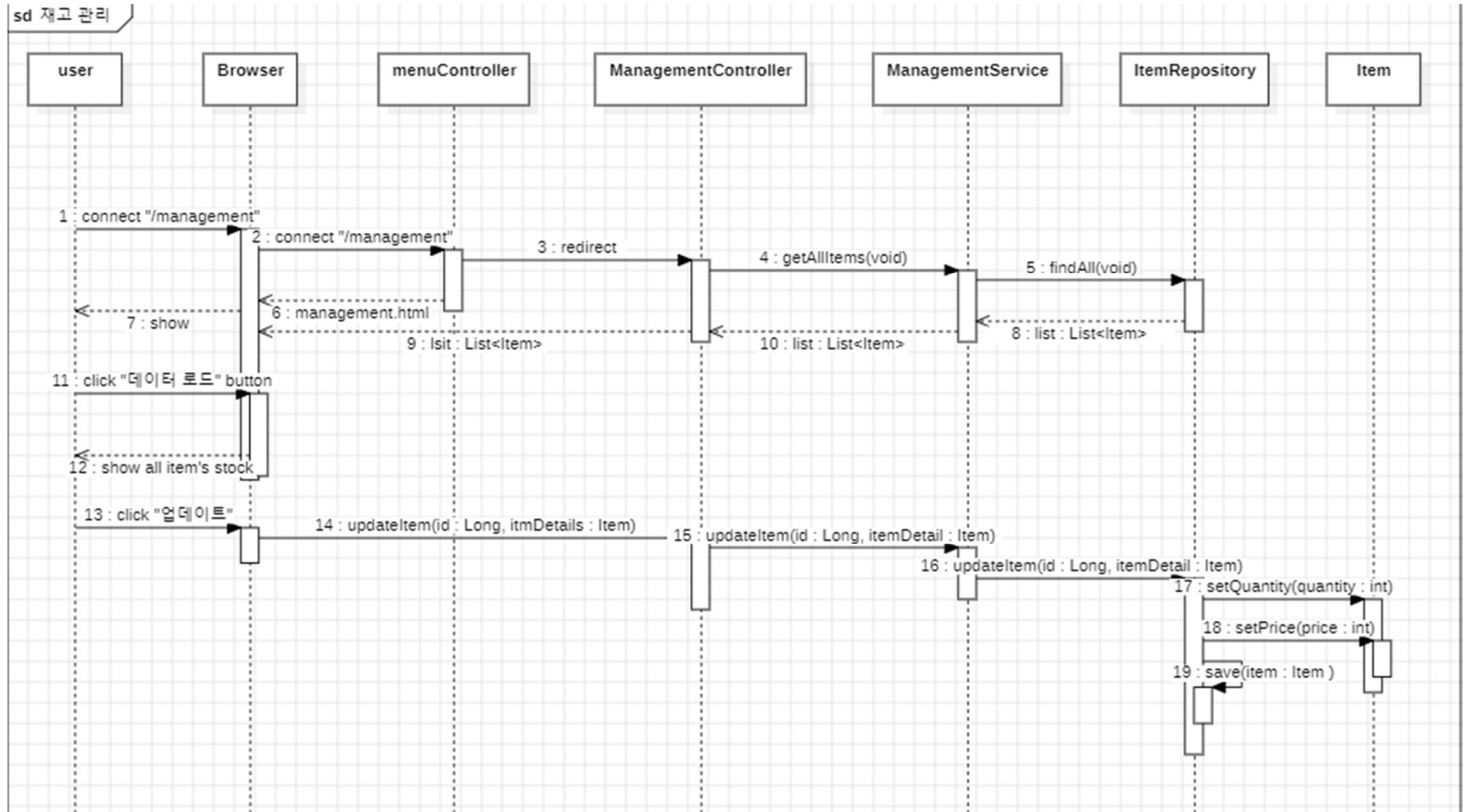
# 변경 후



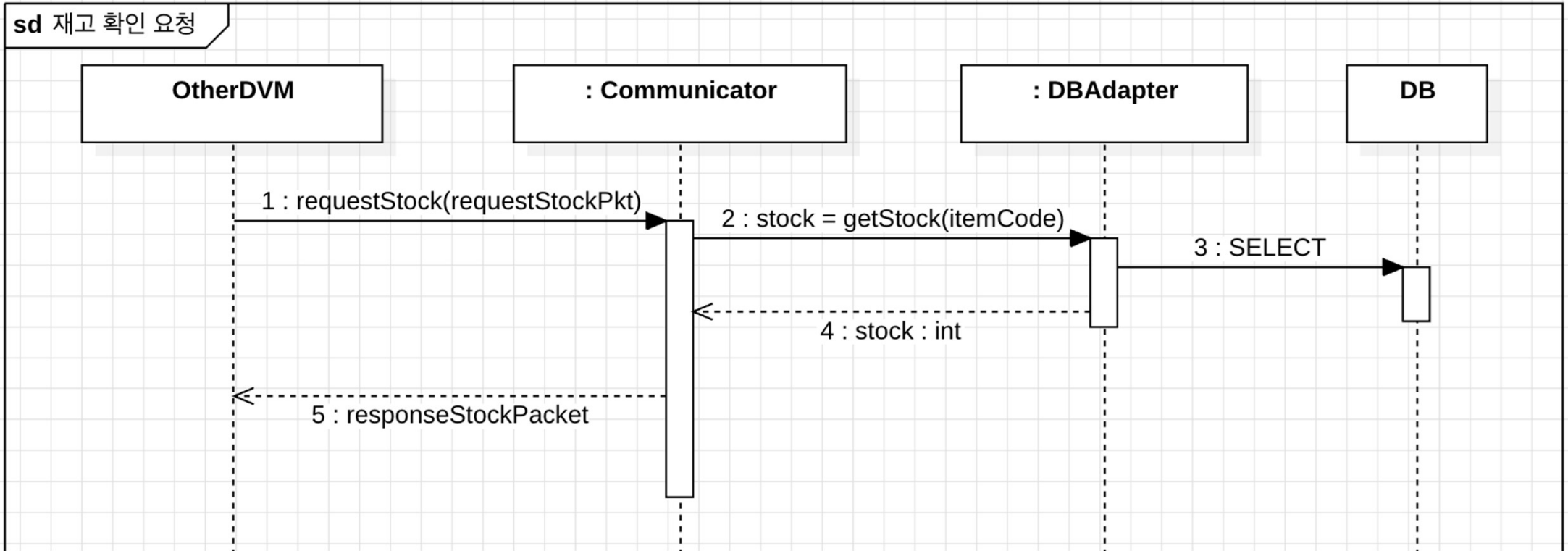
# 변경 전



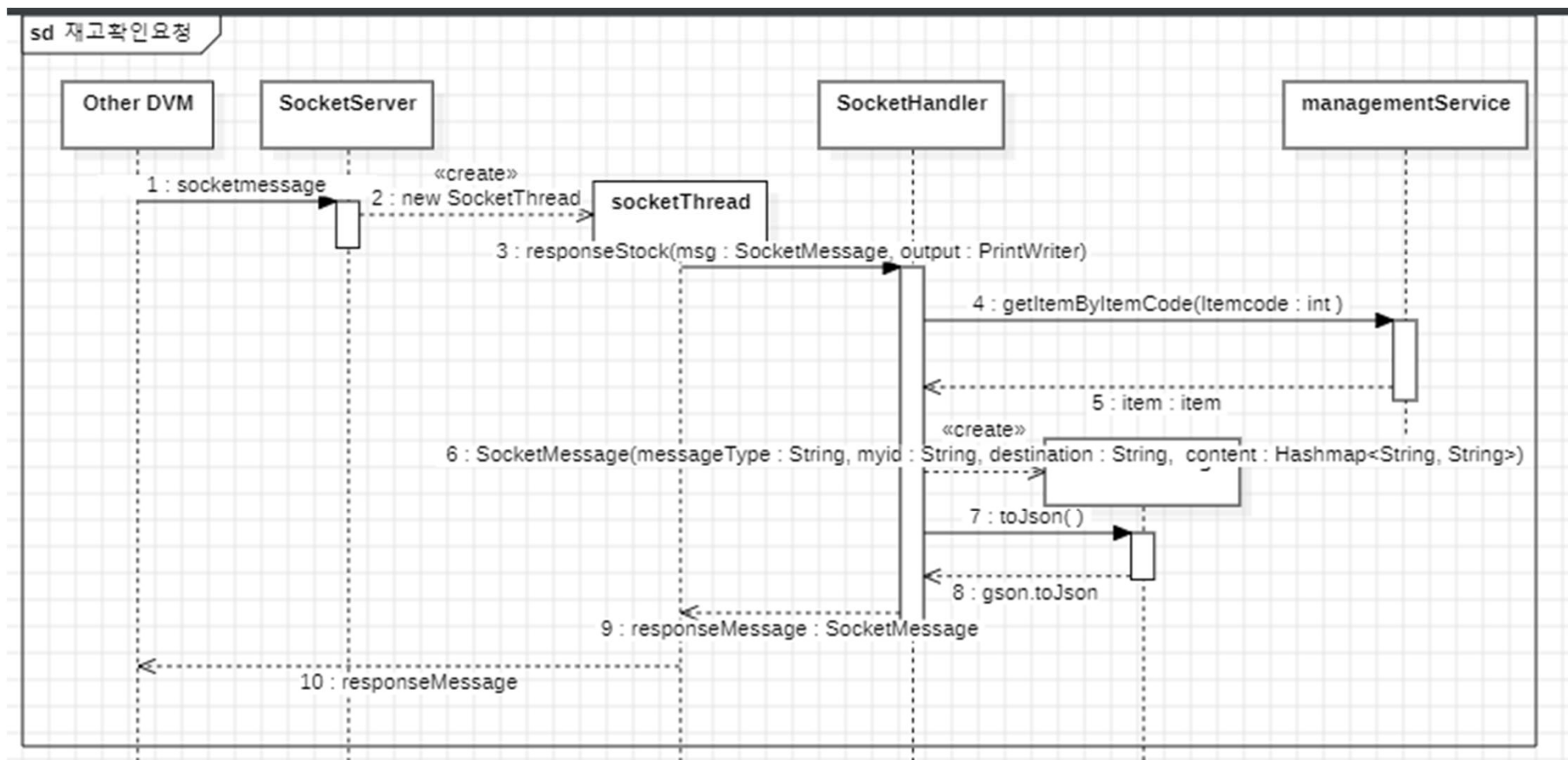
# 변경 후



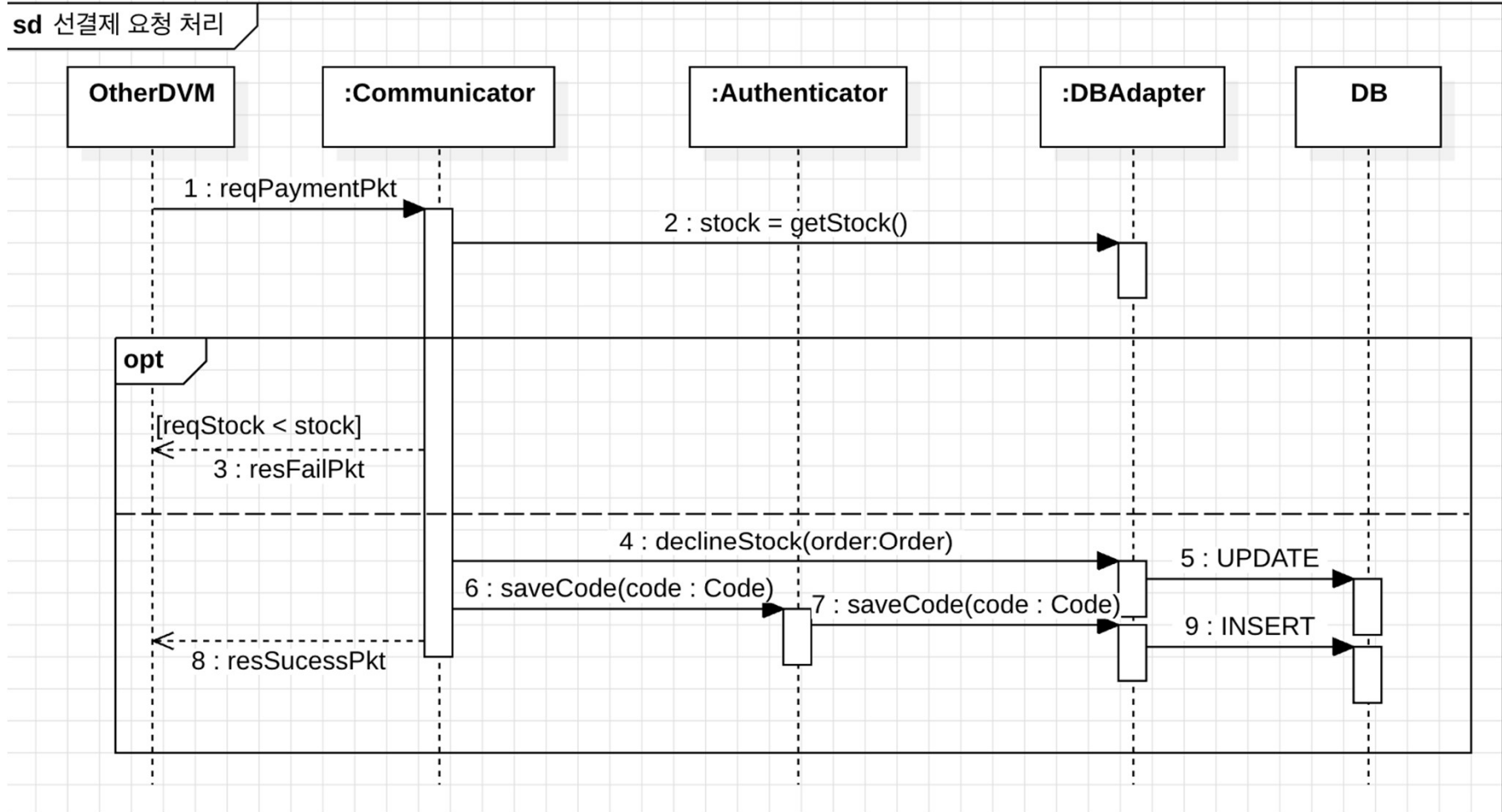
# 변경 전



# 변경 후

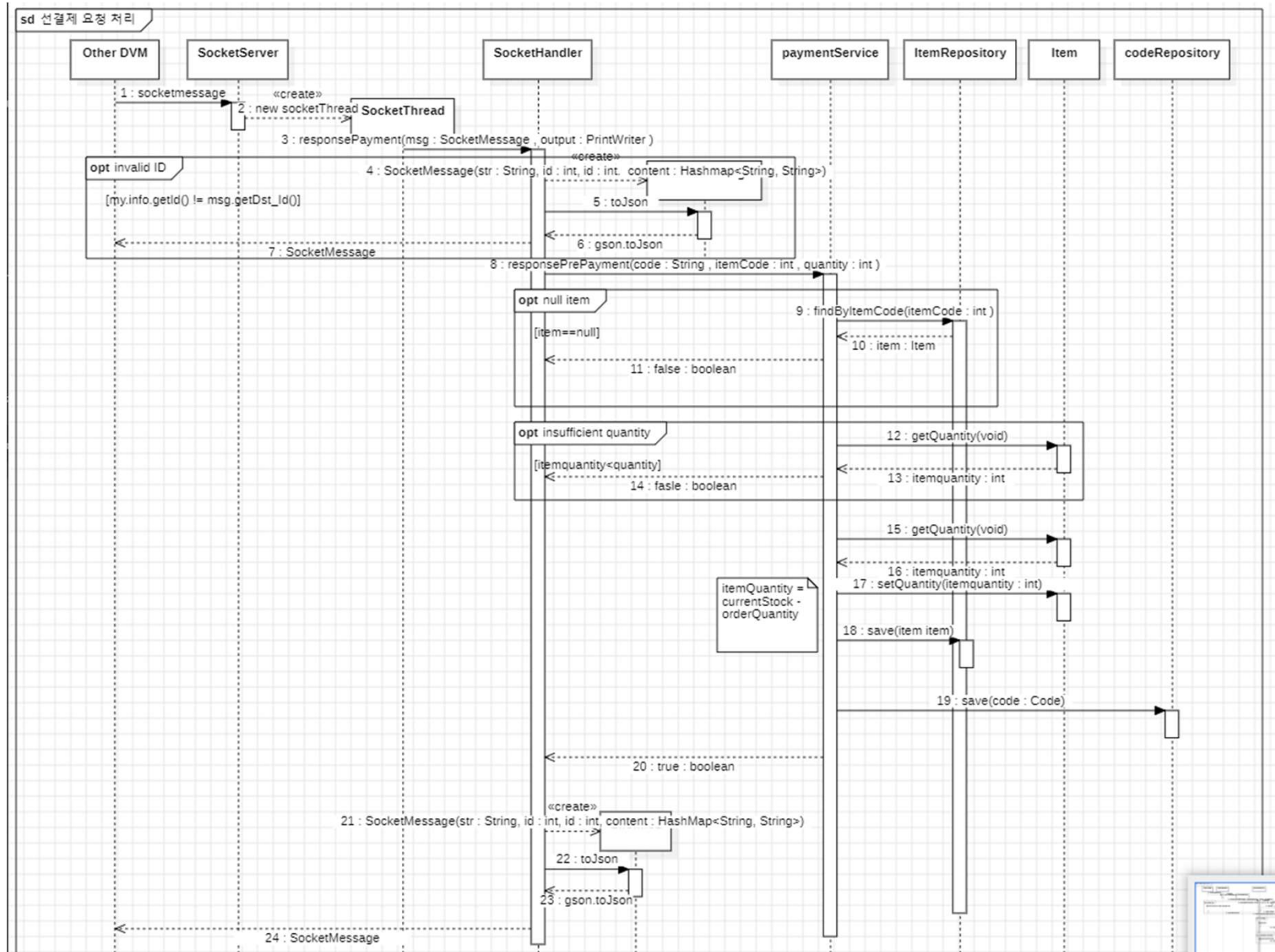


# 변경 전



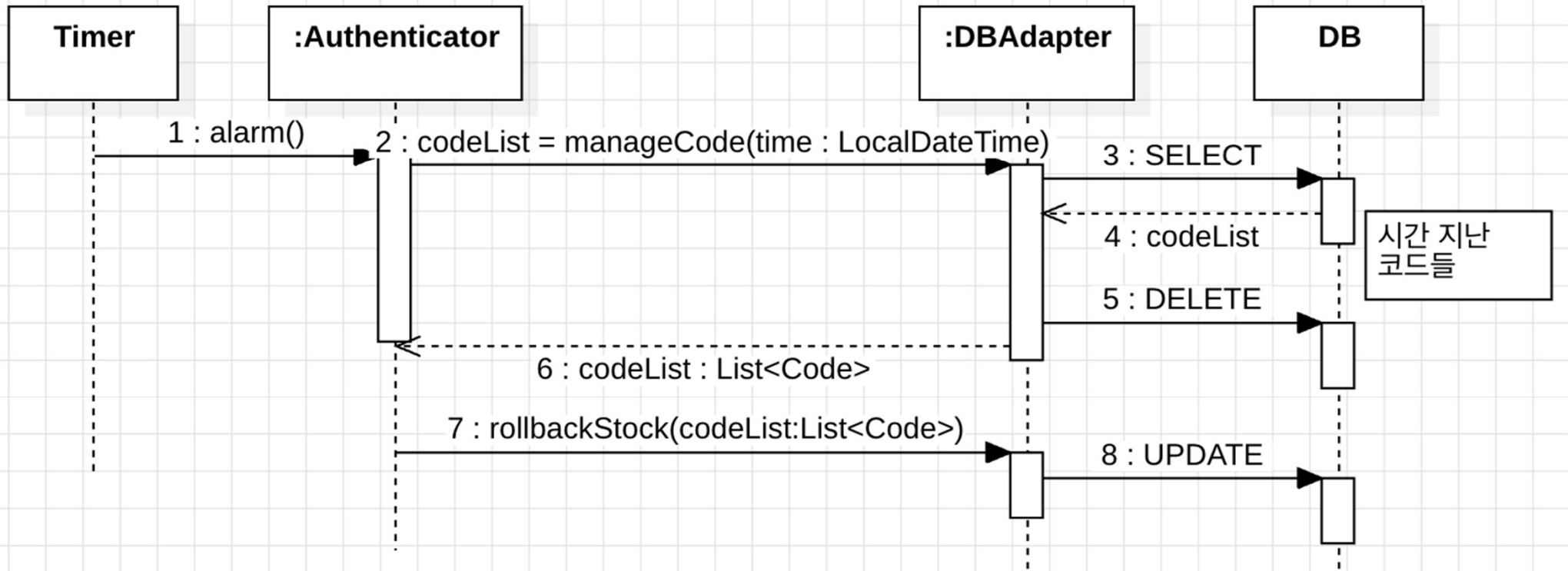


# 변경 후

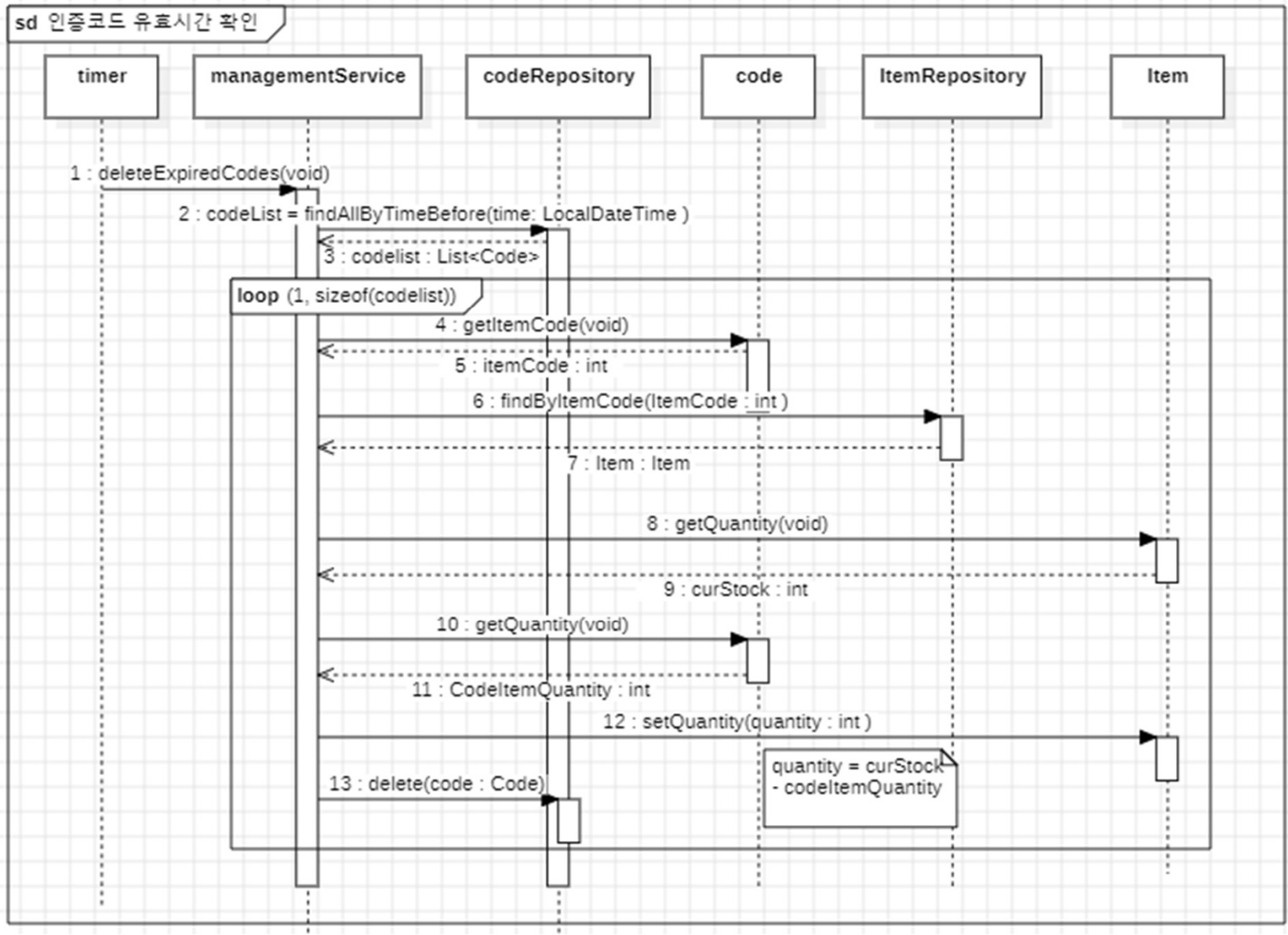


# 변경 전

sd 인증코드 유효 시간 확인



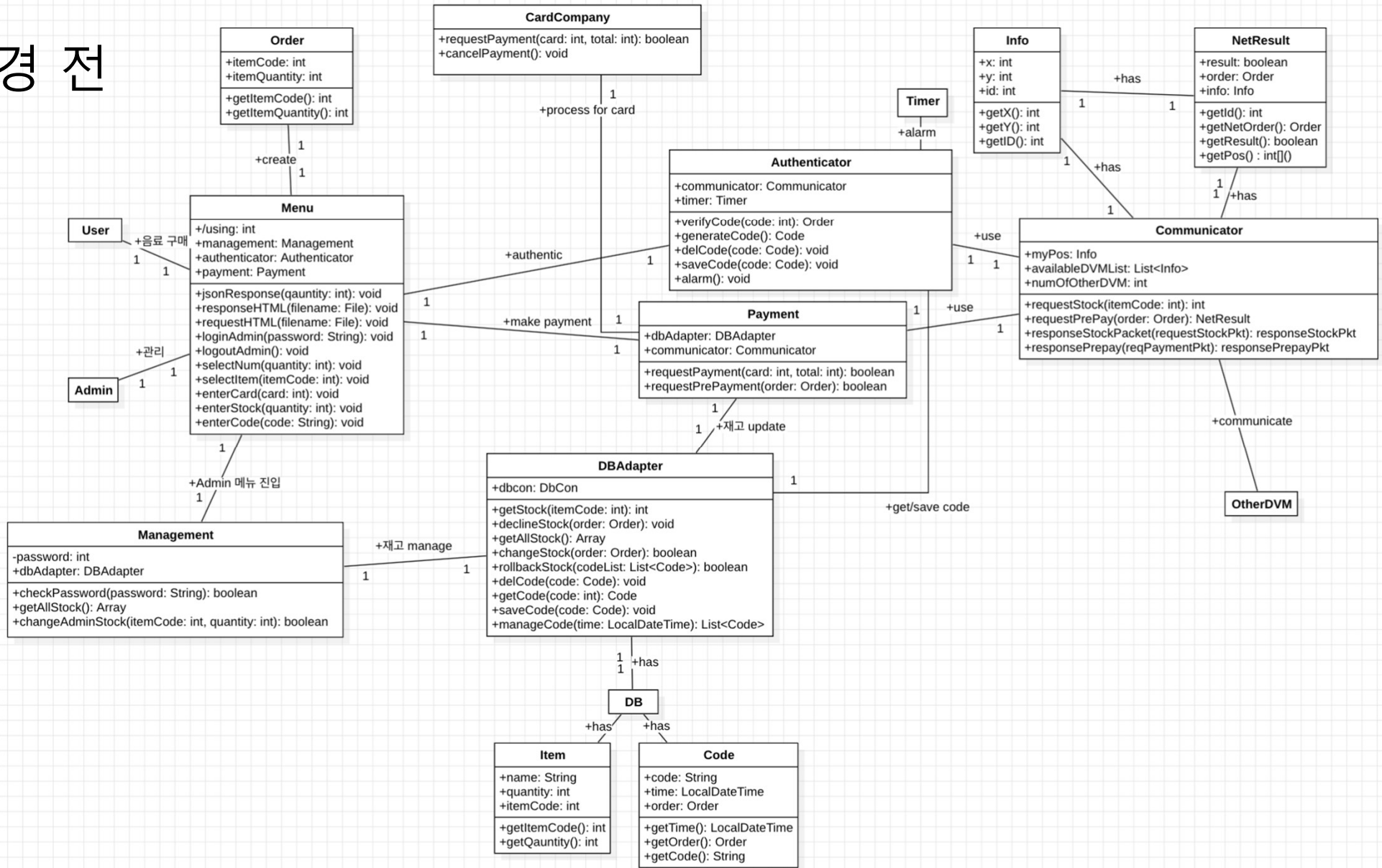
# 변경 후



# Class Diagrams

## 에서 변경된 부분

# 변경 전



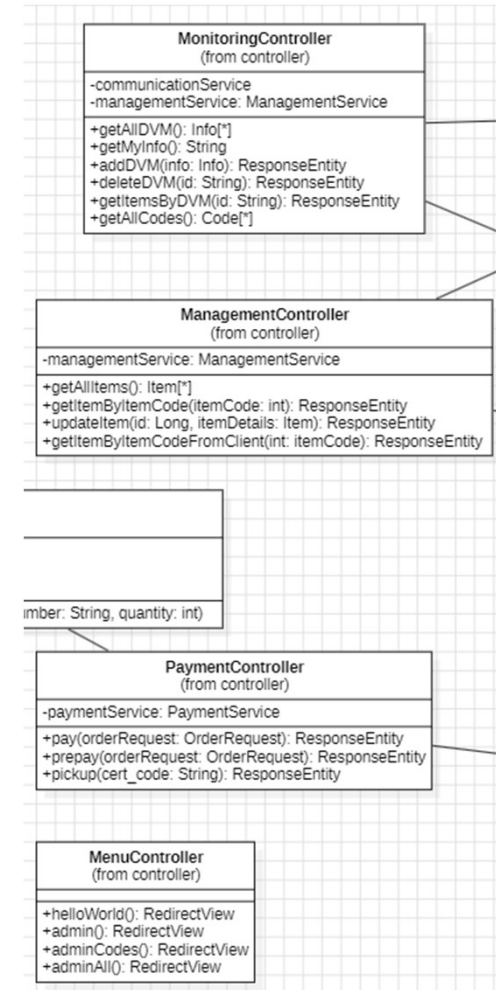
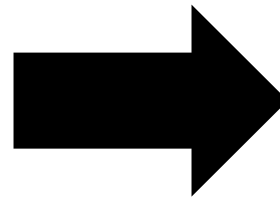
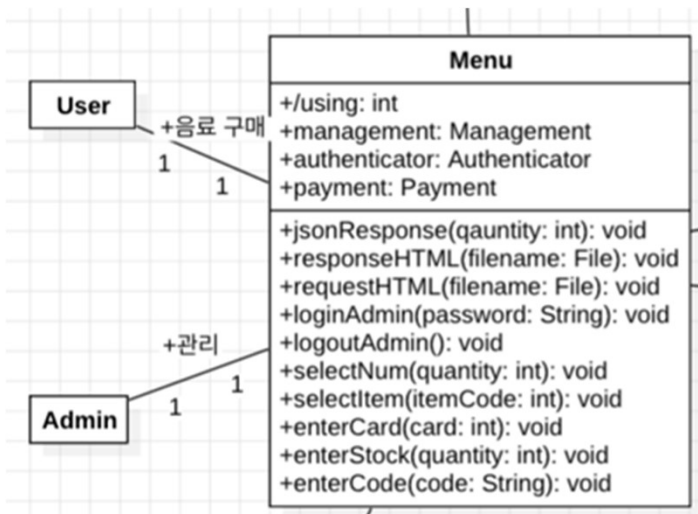


# 크게 변경된 부분 - 메뉴(컨트롤러)

변경 후

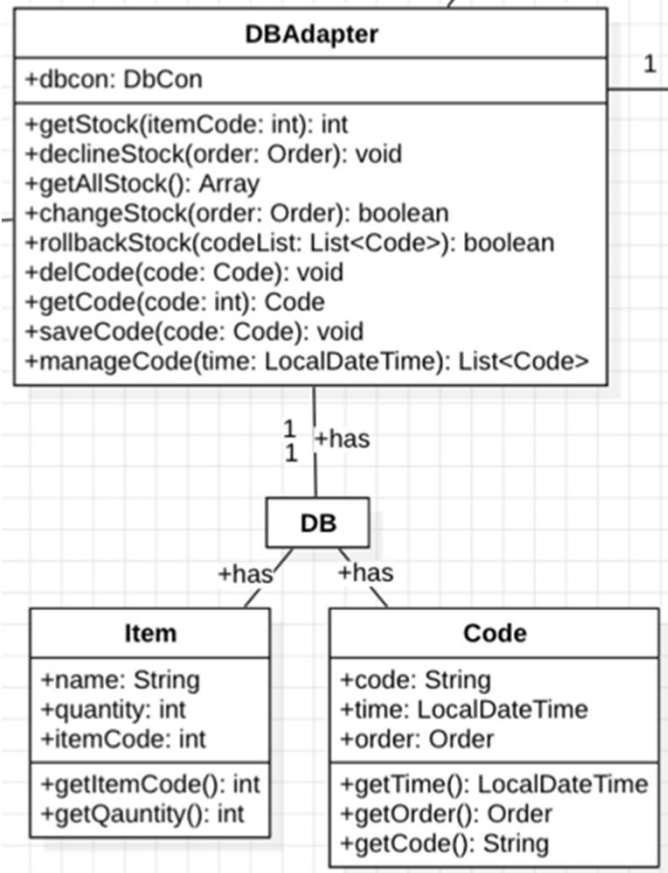
UI (REST 통신) 관련 기능의 수의 예측 실패

변경 전

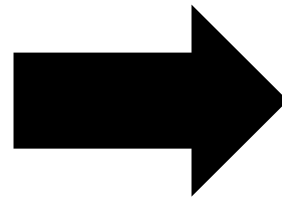


# 크게 변경된 부분 - DB Adapter

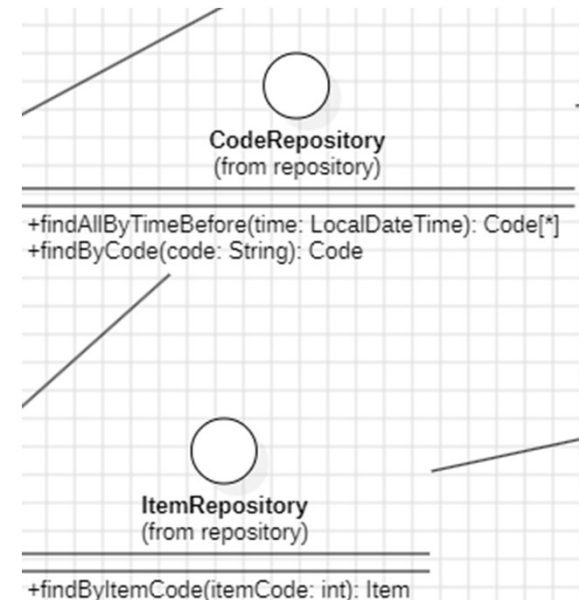
변경 전



구현 방식 변경  
(to JPA)에 따른  
설계 변경



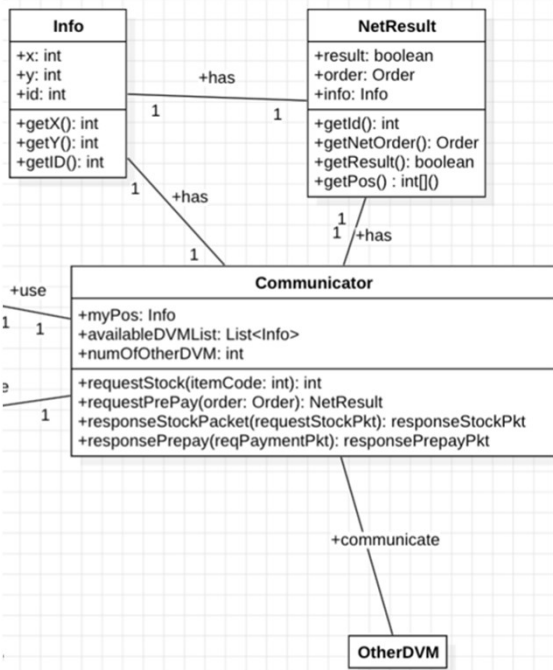
변경 후





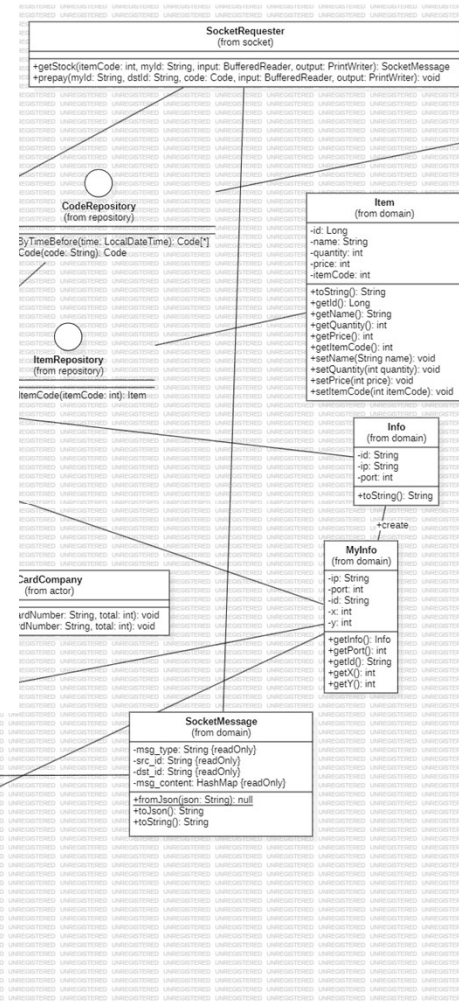
# 크게 변경된 부분 - 통신(Socket)

변경 전



통신 관련  
(Socket) 기능의  
수를 예측 실패

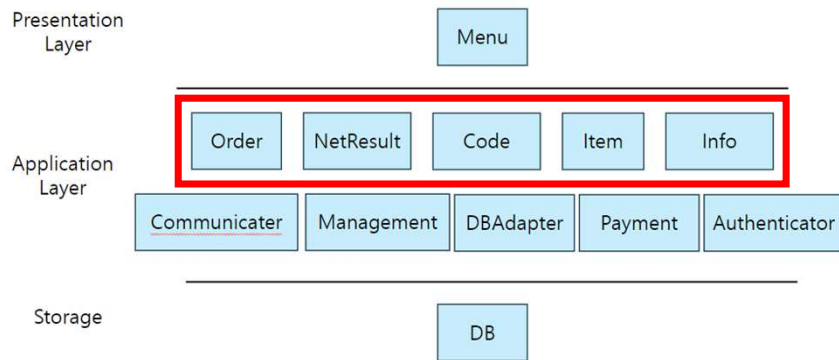
변경 후



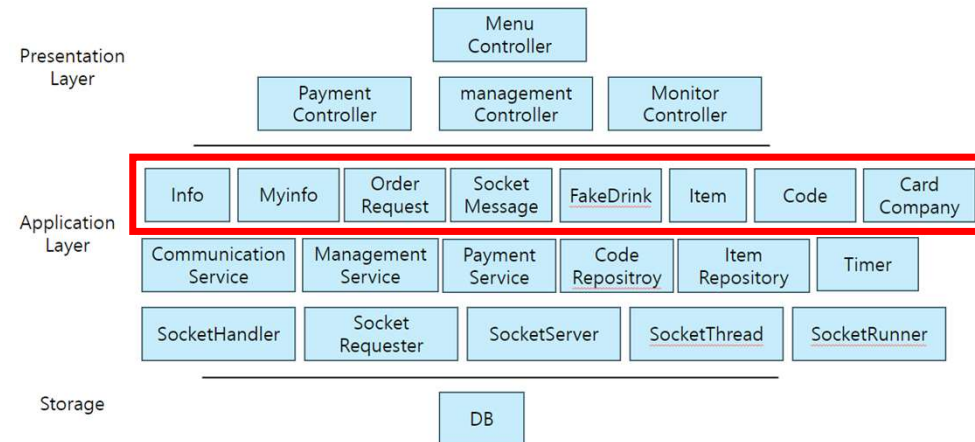
# 크게 변경된 부분 - Domain

객체간 통신을  
하면서 필요한  
Class의 부족

변경 전



변경 후



# 구현 시 예상보다 어려웠던 점

## - 변경된 계획에 따른 재설계

▶ 기존에는 프레임워크를 사용하지 않고 개발 예정이었지만, 수월한 개발을 위해 Spring을 사용하는 것으로 계획 변경

▶ 부족한 재설계 시간에 따른 개발과 설계 간의 괴리

## - 부족한 설계에 따른 개발

▶ 설계 미스 & 애매한 부분

▶ 또 다시 설계로 돌아와 다시 설계

# 구현 시 예상보다 쉬웠던 점

## - 그래도 필요한 기능은 뽑았다

▶ Use Case와 Sequence Diagram을 이용하여, 필요한 기능과 또 그 기능을 위한 기능을 설계 과정에서 생각하고 개발을 하여, 기능 자체가 추가되는 일은 없었다.

## - 웹으로 구현한 점

▶ GUI에 대한 수고가 줄었다

▶ 프레임워크를 이용해 간편한 REST API의 사용/구현

▶ 이미 구현되어 있는 웹 테스트 기능 (Swagger)을 사용하여 손쉬운 사용

# 객체지향개발방법론의 장/단점

## - 장점

- '객체' 지향 개발로 인한 유지보수 / Unit Test의 수월함
- 미리 Use Case 파악, Sequence Diagram 설계, Class Diagram 설계를 통해 전체적인 그림은 다들 숙지 후, 개발에 임해 팀원간 충돌이 적었음

## - 단점

- 미숙련자에게는 오히려 더 큰 수고 ( 재설계 )
- 시간의 소요

# 객체지향개발방법론에 대한 소감

- 팀원 K : 시간적 압박이 컸다. 설계에 대해 제대로 배우고 해보고 싶다.
- 팀원 S : Client driven risk, Architecture driven risk를 실습 중 느끼고, 설계의 중요성을 깨달았다.
- 팀원 S : 팀 단위 대규모 작업을 할 때에는 설계가 필수적인 것을 느꼈다.
- 팀원 L : 배경지식을 더 쌓고 수업을 들었으면 더 유익한 시간이 되었을 것 같은데 아쉽다.