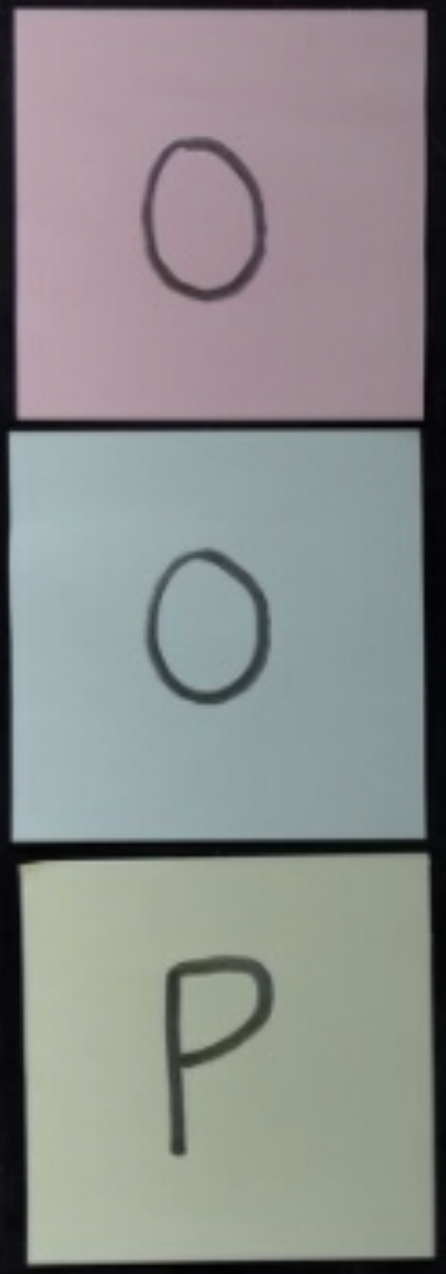


OOPT Final Presentation

객체지향개발방법론

Team[6]

201711374 권규형
201814119 문지영
201911167 김현정
202011370 조석래

CONTENTS

Start

OOPT 1000 [Planning]

2030

2040

2050

2060

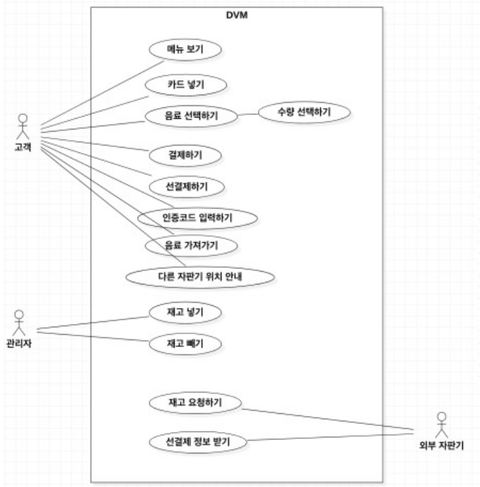
Final

Define Requirements (FR, NFR)

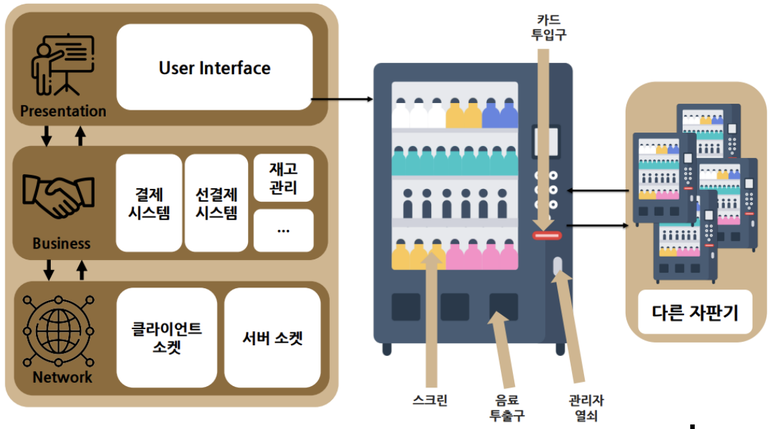
Ref. #	Functional Requirement	Category
R1.1	카드 정보 읽기	evident
R1.2	인증코드 확인하기	evident
R2.1	음료 선택받기	evident
R3.1	결제 진행하기	evident
R4.1	대상 자판기 선정하기	hidden
R4.2	선결제 진행하기	evident
R5.1	음료 판매 정보 응답하기	evident
R5.2	재고 정보 응답하기	evident
R5.3	선 결제 정보 저장하기	evident
R6.1	관리자 모드 진입하기	evident
R6.2	재고 변경하기	evident
R6.3	관리자 모드 종료하기	evident

Define Business Usecase

Ref. #	FR	#	Use Case	Actor	Description
R1.1	카드 정보 읽기	1	카드 넣기	고객	사용자가 카드를 자판기 안에 넣으며 자판기는 이 카드가 유효한 지 체크한다. 유효하다면 기본 상태에서 자판기를 활성화한다
R1.2	인증코드 확인하기	2	인증코드 입력하기	고객	사용자는 인증코드를 입력하고 자판기는 인증코드가 유효한지 검사한다.
R2.1	음료 선택받기	3	음료 선택하기	고객	고객이 구매할 음료와 개수를 입력한다.



Define Draft System Architecture

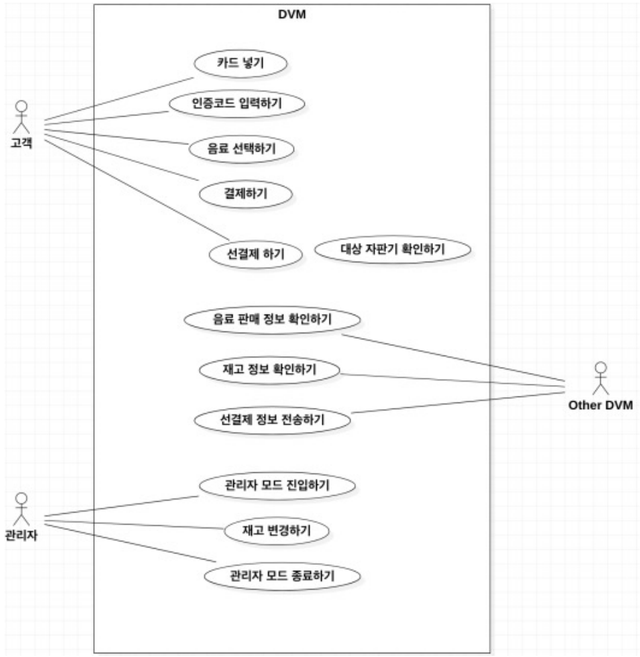


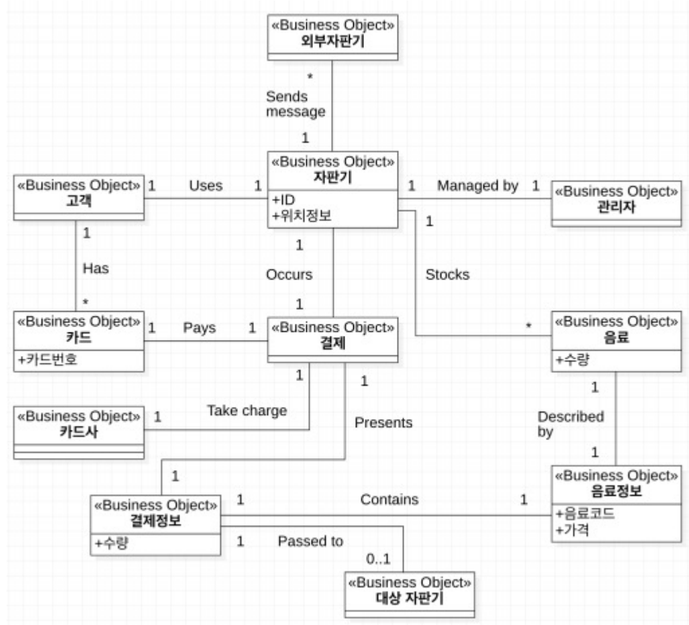
Define Essential Use Cases

Use Case	3. 음료 선택하기
Actor	고객
Purpose	구매할 음료를 선택한다.
Overview	고객이 구매할 음료와 개수를 입력한다.
Type	evident
Cross Reference	Functions: R1.1, R2.1
	Use Case: 1. 카드 넣기
Pre-Requisites	유효한 카드를 넣었다.
Typical	(A): Actor, (S): System 1. (A) 고객이 구매할 음료와 수량을 입력한다. (E1) 2. (S) 현 자판기의 재고와 고객이 구매할 음료의 수량을 비교한다. (E2) 3. (S) 고객에게 결제 여부를 묻는다.
Alter	
Except	(E1): 잘못된 음료를 선택한 경우 잘못됐음을 알리고 기본 상태로 돌아간다. (E2): 재고가 충분하지 않다면 대상 자판기를 탐색한다.

Define Domain Model

User | Item | DVM

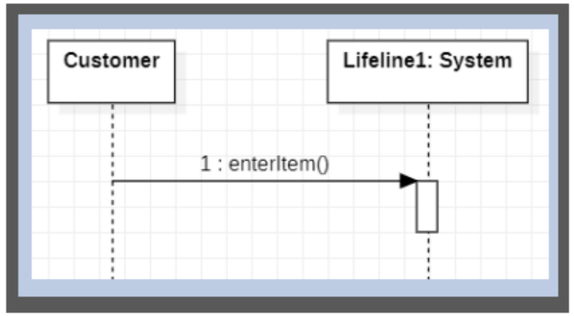




Define System Sequence Diagrams

Use Case : 3. 음료 선택하기
Typical Course of Events

1. 고객이 구매할 음료와 수량을 입력한다



Refine System Test Case

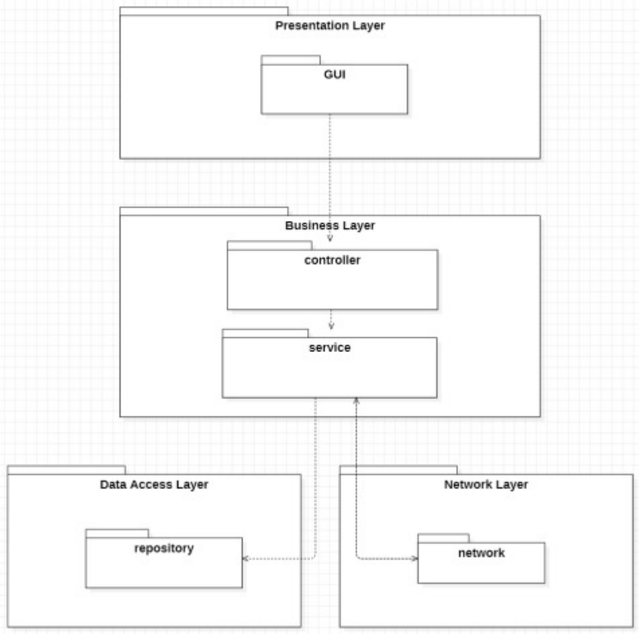
Test case	No.	Condition	Result	Use case
1. 카드 인식 기능	1	유효한 카드를 입력한다.	- 자판기가 활성화되는 것을 확인한다.	1. 카드 넣기
	2	유효하지 않은 카드를 입력한다.	- 자판기가 활성화되지 않는 것을 확인한다.	
2. 인증코드 기능	3	유효한 인증코드를 입력한다.	- 알맞은 음료가 제공되는 것을 확인한다.	2. 인증코드 입력하기
	4	존재하지 않는 인증코드를 입력한다.	- 존재하지 않은 인증코드임을 알려주는 지 확인한다.	
	5	잘못된 선결제가 일어난 인증코드를 입력한다.	- 선결제가 일어났지만 재고가 없다는 사실을 알려주는 지 확인한다.	
	6	자리수가 맞지 않는 인증코드 입력한다.	- 인증코드 형식이 다르다는 메시지를 띄운다.	
3. 음료 선택 기능	7	유효한 음료와 현 자판기에 재고가 있는 만큼의 수량을 입력한다.	- 결제 여부를 묻는 지 확인한다.	3. 음료 선택하기
	8	유효한 음료와 현 자판기에 없는 만큼의 수량을 입력한다.	- 대상 자판기를 찾아보는 지 확인한다.	
	9	존재하지 않는 음료를 입력한다.	- 음료 선택 에러 메시지를 띄운 후 기본 상태로 돌아가는 지 확인한다.	
	10	(1 보다 작은 수량) 또는 (999보다 큰 수량)을 입력한다.	- 음료 선택 에러 메시지를 띄운 후 기본 상태로 돌아가는 지 확인한다.	
4. 결제 기능	11	자판기 재고를 넘지 않는 개수의 음료를 잔액이 충분한 카드를 사용하여 결제한다.	- 결제가 잘 진행되는 지 확인한다. - 현 자판기의 상품 정보를 업데이트 하는 지 확인한다. - 상품을 배출하는 지 확인한다. - 기본 상태로 돌아가는 지 확인한다.	4. 결제하기
	12	잔액이 부족한 카드로 결제한다.	- 결제가 진행되지 않는 지 확인한다. - 잔액 부족 에러 메시지를 띄우는 지 확인한다.	
	13	결제 도중 재고가 변경되어 수량이 부족해진다.	- 결제가 진행되지 않는 지 확인한다. - 재고 부족 에러 메시지를 띄우는 지 확인한다.	

OOPT 2040 [Design]

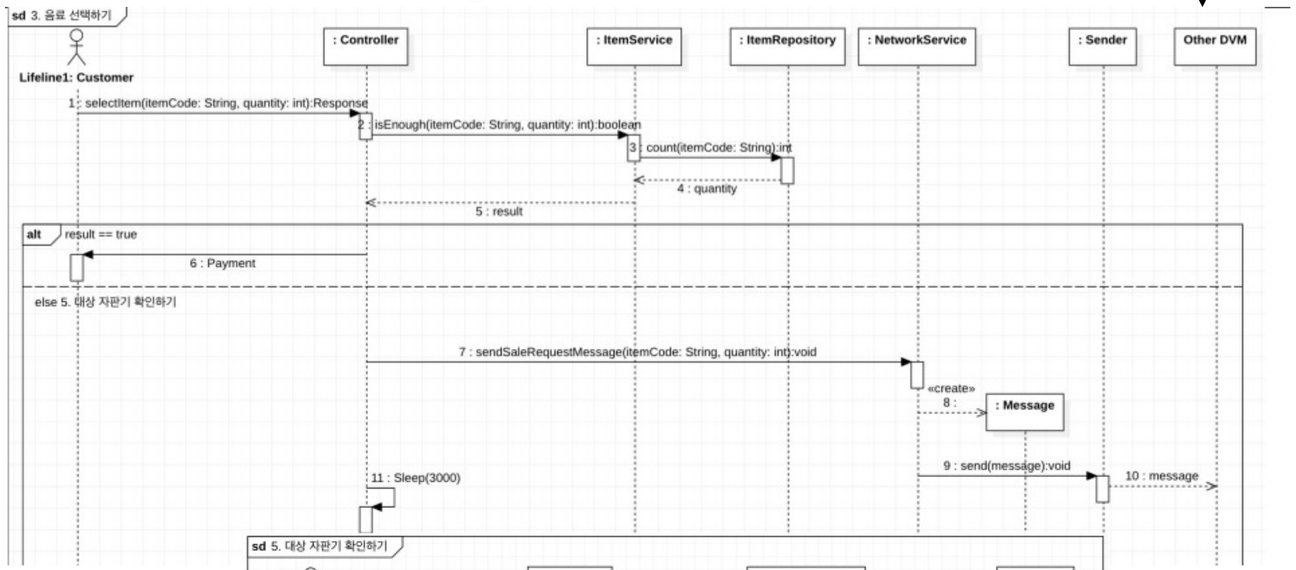
Define Reports, UI, and StoryBoards

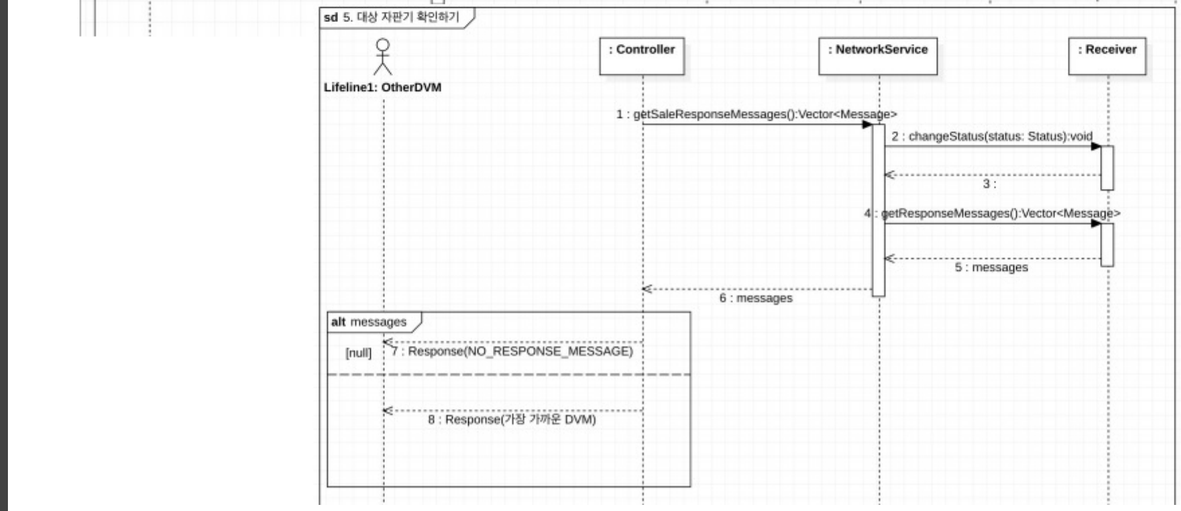
Design Real Use Case

Refine System Architecture

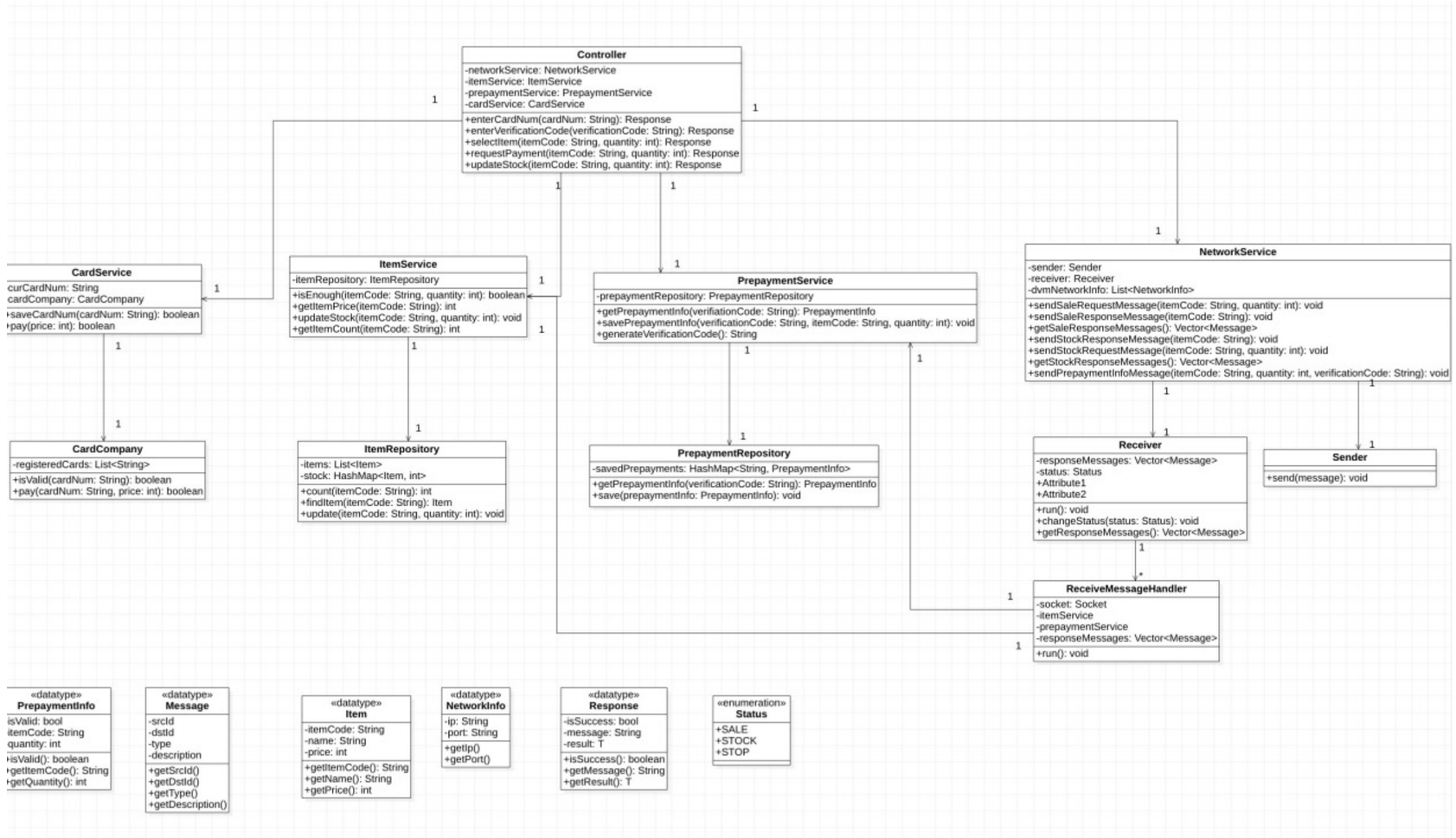


Define Interaction Diagrams





Define Design Class Diagram



OOPT 2050 [Construct]

Implement

환경 : Intellij + Git

빌드 : Gradle

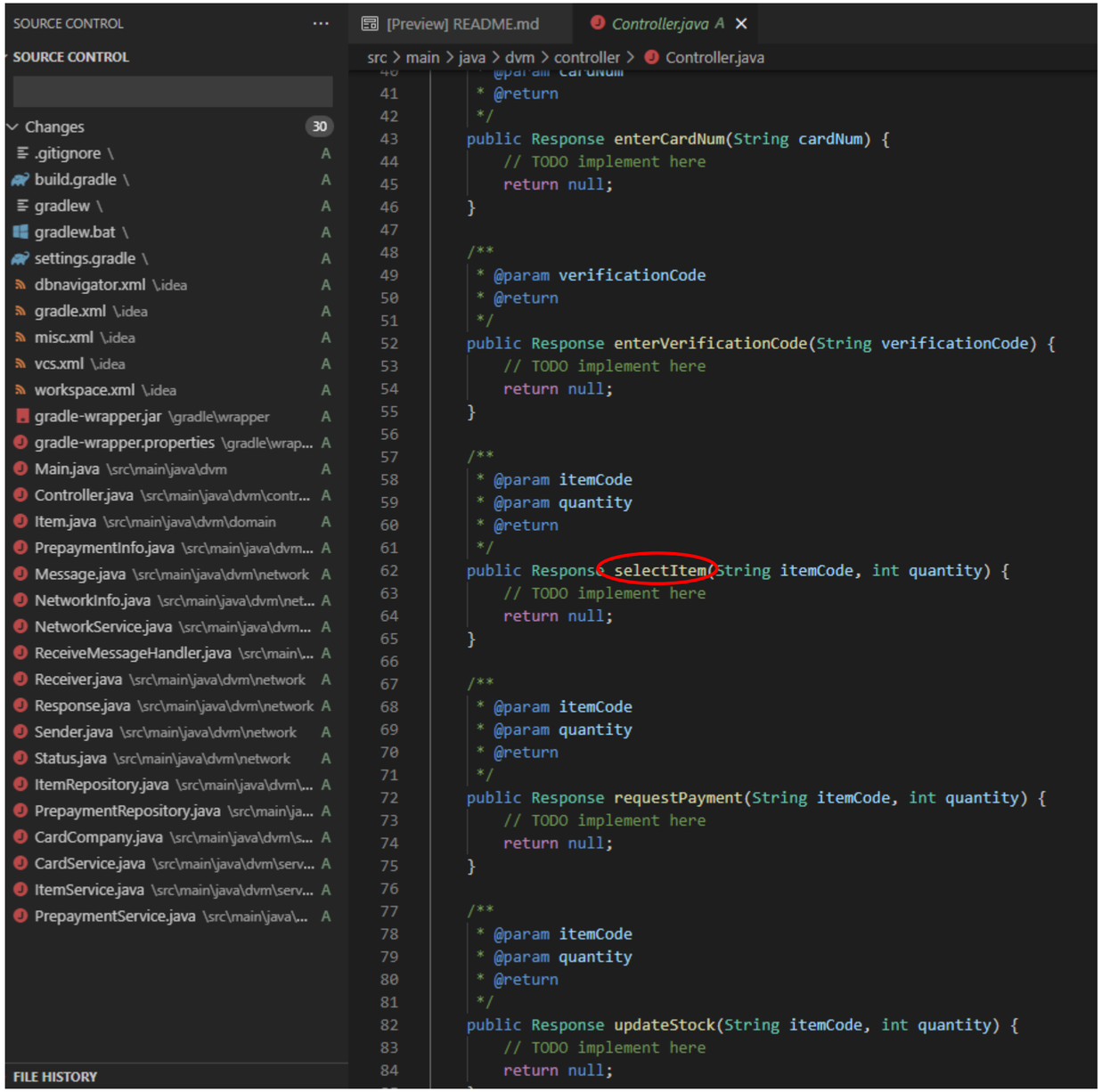
리소스 : Properties (세팅 간편화)

구현 : 객체지향

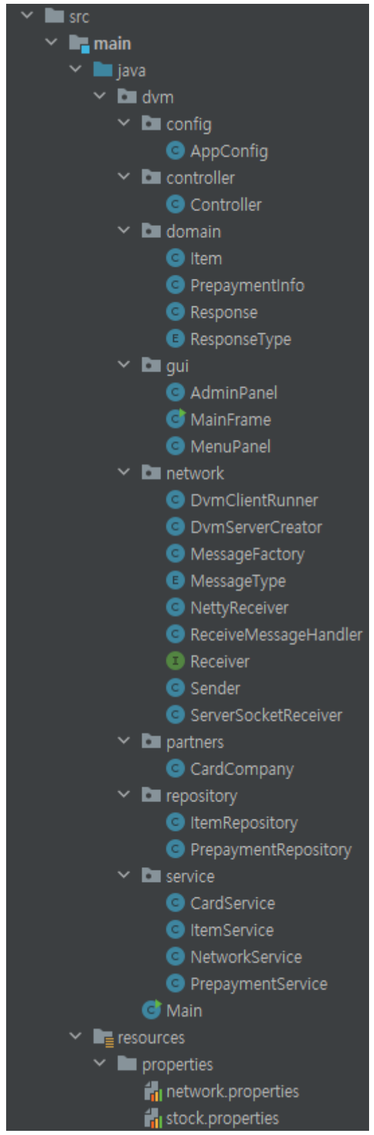
SkeletonCode

From StarUML Class Diagram

\+ 199 commits



OOPT 2050 [Construct]




```java
public Response<Message> selectItem(String itemCode, int quantity) {
        boolean result = itemService.isEnough(itemCode, quantity);
        if (result) {
            return new Response<>(true, SELECTION_OK);
        }
        networkService.sendSaleRequestMessage(itemCode, quantity);
        try {
            Thread.sleep(5000);
            Message message = networkService.getSaleResponseMessage(itemCode);
            if (message == null) {
                logger.warning("Controller | SaleResponseMessage 없음 | 받은 SaleResponseMessage가
없습니다.");
                return new Response<>(false, NO_RESPONSE_MESSAGE);
            }
            logger.info("Controller | SaleResponseMessage 받음 | from " + message.getSrcId() + " |
보유 수량: " + message.getMsgDescription().getItemNum());
            return new Response<>(true, RESPONSE_OK, message);
        } catch (Exception e) {
            return new Response<>(false, NO_RESPONSE_MESSAGE);
        }
    }
```

Unit Test

```java
class PrepaymentServiceTest {
    @Test
    void savePrepaymentInfo() {
        String vCode = "vCode2";
        String iCode = "02";
        int quantity = 10;
        boolean valid = false;

        prepaymentService.savePrepaymentInfo(itemService, vCode, iCode, quantity);
        PrepaymentInfo saveInfo = prepaymentService.getPrepaymentInfo(vCode);

        assertEquals(saveInfo.getItemCode(), iCode);
        assertEquals(saveInfo.getQuantity(), quantity);
        assertEquals(saveInfo.isValid(), valid);
    }

    @Test
    void generateVerificationCode() {
        for (int i = 0; i < 100; i++) {
            String code = prepaymentService.generateVerificationCode();
            assertTrue(code.matches("^.*[0-9].*$"));
            assertTrue(code.matches("^.*[a-z].*$"));
            assertFalse(code.matches("^.*[A-Z].*$"));
        }
    }
}
```

System Test

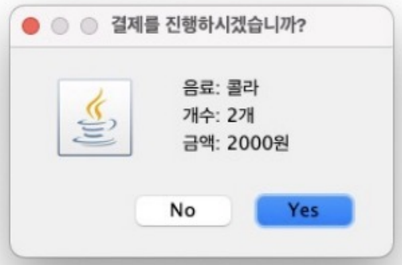
Testcase	#	조건	기대 결과	Usecase
3. 음료 선택 기능	6	유효한 음료와 현 자판기에 재고가 있는 만큼의 수량을 입력한다.	결제 여부를 묻는 지 확인한다	3. 음료 선택하기
	7	유효한 음료와 현 자판기에 없는 만큼의 수량을 입력한다.	대상 자판기를 찾아보는 지 확인한다	
	8	(1보다 작은 수량)또는 (999보다 큰 수량)을 입력한다.	1. 초기 상태(0개)일 때는 결제 버튼을 눌러도 결제가 안된다. 2. 999 에선 +버튼이 안눌리는 것을 확인	

1. 현재 재고 정보. 01번(콜라)이 100개 있음

```
------------------현재 재고 정보-------------------
item code: 01 | count: 100
item code: 02 | count: 0
item code: 05 | count: 0
item code: 06 | count: 0
item code: 18 | count: 0
item code: 19 | count: 0
item code: 20 | count: 10
--------------------------------------------------
```

2. 존재하는 만큼의 수량(2개)으로 콜라 선택

3. 정상적으로 결제 여부를 묻는다.



OOPT 2060 [Testing]

Testcase	#	조건	기대 결과	Usecase
3. 음료 선택 기능	6	유효한 음료와 현 자판기에 재고가 있는 만큼의 수량을 입력한다.	결제 여부를 묻는 지 확인한다	3. 음료 선택하기
	7	유효한 음료와 현 자판기에 없는 만큼의 수량을 입력한다.	대상 자판기를 찾아보는 지 확인한다	
	8	(1보다 작은 수량)또는 (999보다 큰 수량)을 입력한다.	1. 초기 상태(0개)일 때는 결제 버튼을 눌러도 결제가 안된다. 2. 999 에선 +버튼이 안눌리는 것을 확인	

1. 현재 재고 정보

```
------------------현재 재고 정보--------------------
item code: 01 | count: 100
item code: 02 | count: 0
item code: 05 | count: 0
item code: 06 | count: 0
item code: 18 | count: 0
item code: 19 | count: 500
item code: 20 | count: 500
-----------------------------------------------
```

2. 현재 재고가 없는 사이다(02번) 구매 시도

3. 대상 자판기를 찾아본다.

```
receiver 상태 수정 | SalesCheckResponse을 기다리는 중
5월 26, 2022 2:18:11 오후 dvm.service.ItemService isEnough
INFO: 확인 요청 수량: 1 | 현재 재고: 0
5월 26, 2022 2:18:11 오후 dvm.network.Sender send
INFO: 전송 시도 | to Team4
5월 26, 2022 2:18:11 오후 dvm.network.Sender send
INFO: 전송 시도 | to Team3
channel Registered
channel Registered
channel Active
channel Active
sendMessage:{"srcId":"Team1","dstID":"0","msgType":"SalesCheckRequest","msgDescription":{"itemCode":"02","itemNum":1,"dvmXCoord":0,"dvmYCoord":0}}
sendMessage:{"srcId":"Team1","dstID":"0","msgType":"SalesCheckRequest","msgDescription":{"itemCode":"02","itemNum":1,"dvmXCoord":0,"dvmYCoord":0}}
channel UnRegistered
channel UnRegistered
5월 26, 2022 2:18:11 오후 dvm.network.DvmClientRunner run
INFO: 메세지 전달 완료 | to 0 | SalesCheckRequest |
5월 26, 2022 2:18:11 오후 dvm.network.DvmClientRunner run
INFO: 메세지 전달 완료 | to 0 | SalesCheckRequest |
Channel Registered
Channel is activated
[Model.Message@787f936b]
Channel is inactivated
Channel Unregistered
5월 26, 2022 2:18:11 오후 dvm.network.NettyReceiver run
INFO: SALE_RESPONSE type 메세지 추가 from Team3
```

OOPT 2060 [Testing]

Testcase	#	조건	기대 결과	Usecase
3. 음료 선택 기능	6	유효한 음료와 현 자판기에 재고가 있는 만큼의 수량을 입력한다.	결제 여부를 묻는 지 확인한다	3. 음료 선택하기
	7	유효한 음료와 현 자판기에 없는 만큼의 수량을 입력한다.	대상 자판기를 찾아보는 지 확인한다	
	8	(1보다 작은 수량)또는 (999보다 큰 수량)을 입력한다.	1. 초기 상태(0개)일 때는 결제 버튼을 눌러도 결제가 안된다. 2. 999 에선 +버튼이 안눌리는 것을 확인	

1. 0개일때는 결제버튼을 눌러도 결제가 안되고,
-버튼을 눌러도 개수가 줄어들지 않는다.

2. 999에선 +버튼을 눌러도 개수가 늘어나지 않는다.

Final

Traceability Table

Start

2030

2040

2050

2060

Functional Requirements		Use Case		System Operation
1.1	카드 정보 읽기	1	카드번호 입력하기	enterCardNum
1.2	인증코드 확인하기	2	인증코드 입력하기	enterVerificationCode
2.1	음료 선택하기	3	음료 선택하기	selectItem
3.1	결제 진행하기	4	결제하기	
4.1	대상 자판기 선정하기	5	대상 자판기 확인하기	
4.2	선결제 진행하기	6	선결제 하기	
5.1	음료판매 정보 응답하기	7	음료판매 정보 확인하기	
5.2	재고 정보 응답하기	8	재고 정보 확인하기	
5.3	선결제 정보 저장하기	9	선결제 정보 전송하기	
6.1	관리자 모드 진입하기	10	관리자 모드 진입하기	
6.2	재고 변경하기	11	재고 변경하기	
6.3	관리자 모드 종료하기	12	관리자 모드 종료하기	

Method	Class
getAllItems(): Response	Controller
getMyItems(): Response	
getItemCount(itemCode: String): Response	
enterCardNum(cardNum: String): Response	
enterVerificationCode(verificationCode: String): Response	
selectItem(itemCode: String, quantity: int): Response	
requestPayment(itemCode: String, quantity: int): Response	
requestPrePayment(dstId:String,itemCode:String,quanti:int):Response	
updateStock(itemCode: String, quantity: int): Response	
getPrepaymentInfo(String verifiationCode): PrepaymentInfo	PrepaymentService
savePrepaymentInfo(String verificationCode, String itemCode, int quantity): void	
generateVerificationCode(): String	
getPrepaymentInfo(String verificationCode): PrepaymentInfo	PrepaymentRepository
save(PrepaymentInfo prepaymentInfo): void	
isValid(): boolean	PrepaymentInfo
getItemCode(): String	
getQuantity(): int	
saveCardNum(cardNum: String): Boolean	CardService
pay(price: int): Boolean	
isVaild(cardNum: String): Boolean	CardCompany
pay(cardNum: String, price: int): boolean	
isEnough(itemCode: String, quantity: int): Boolean	ItemService
getItemPrice(itemCode: String, quantity: int): boolean	
updateStock(itemCode: String, quantity: int): void	
getItemCount(itemCode: String): int	
getItems(): List	
getMyItems(): List	
getItemCount(): int	
count(itemCode:String): int	ItemRepository
findItem(itemCode:String): Item	
update(itemCode:String, quantity:int): void	

Final

Traceability table

Start

2030

2040

2050

2060

Functional Requirements		Use Case		System Operation
1.1	카드 정보 읽기	1	카드번호 입력하기	
1.2	인증코드 확인하기	2	인증코드 입력하기	
2.1	음료 선택하기	3	음료 선택하기	
3.1	결제 진행하기	4	결제하기	reqPayment
4.1	대상 자판기 선정하기	5	대상 자판기 확인하기	reqPrepayment
4.2	선결제 진행하기	6	선결제 하기	
5.1	음료판매 정보 응답하기	7	음료판매 정보 확인하기	
5.2	재고 정보 응답하기	8	재고 정보 확인하기	
5.3	선결제 정보 저장하기	9	선결제 정보 전송하기	
6.1	관리자 모드 진입하기	10	관리자 모드 진입하기	
6.2	재고 변경하기	11	재고 변경하기	
6.3	관리자 모드 종료하기	12	관리자 모드 종료하기	

Method	Class
getAllItems(): Response	Controller
getMyItems(): Response	
getItemCount(itemCode: String): Response	
enterCardNum(cardNum: String): Response	
enterVerificationCode(verificationCode: String): Response	
selectItem(itemCode: String, quantity: int): Response	
requestPayment(itemCode: String, quantity: int): Response	
requestPrepayment(dstId:String,itemCode:String,quantity:int):Response	
updateStock(itemCode: String, quantity: int): Response	
saveCardNum(cardNum: String): boolean	CardService
pay(price: int): boolean	
isVaild(cardNum: String): boolean	CardCompany
pay(cardNum: String, price: int): boolean	
isEnough(itemCode: String, quantity: int): boolean	ItemService
getItemPrice(itemCode: String, quantity: int): boolean	
updateStock(itemCode: String, quantity: int): void	
getItemCount(itemCode: String): int	
getItems(): List	
getMyItems(): List	
getItemCount(): int	
count(itemCode:String): int	ItemRepository
findItem(itemCode:String): Item	
update(itemCode:String, quantity:int): void	
sendSaleRequestMessage(itemCode:String, quantity:int): void	NetworkService
sendSaleResponseMessage(itemCode:String): void	
getSaleResponseMessages(): Vector	
sendStockResponseMessage(itemCode:String): void	
sendStockRequestMessage(itemCode:String, quantity:int): void	
getStockResponseMessagesFrom(): Vector	
sendPrepaymentInfoMessage(itemCode:String, quantity:int, verificationCode:String): void	
getMessages(messageType: MessageType): Vector	
changeWaitingMessageType(messageType: MessageType): void	
clearResponseMessages(): void	
send(message: Message(): void	Sender
run(): void	
initDvmsNetWorkInfo(ids: Strings[], ips: Strings[]): void	
printDvmsNetworkInfo(): void	
changeWaitingMessageType(messageType MessageType): void	Receiver
getResponseMessages(): Vector<Message>	
clearResponseMessages(): void	
generateVerificationCode(): String	

Final

Traceability table

Start

2030

2040

2050

2060

Functional Requirements	
1.1	카드 정보 읽기
1.2	인증코드 확인하기
2.1	음료 선택하기
3.1	결제 진행하기
4.1	대상 자판기 선정하기
4.2	선결제 진행하기
5.1	음료판매 정보 응답하기
5.2	재고 정보 응답하기
5.3	선결제 정보 저장하기
6.1	관리자 모드 진입하기
6.2	재고 변경하기
6.3	관리자 모드 종료하기

Use Case	
1	카드번호 입력하기
2	인증코드 입력하기
3	음료 선택하기
4	결제하기
5	대상 자판기 확인하기
6	선결제 하기
7	음료판매 정보 확인하기
8	재고 정보 확인하기
9	선결제 정보 전송하기
10	관리자 모드 진입하기
11	재고 변경하기
12	관리자 모드 종료하기

System Operation
checkItemInfo
checkRepository
enterAdmin
updateStock
exitAdmin

Method	Class
selectItem(itemCode: String, quantity: int): Response<Message>	Controller
requestPayment(itemCode: String, quantity: int): Response<String>	
getItemCount(itemCode: String): Response<Integer>	
getAllItems(): Response<List<Item>>	
enterCardNum(cardNum: String): Response<String>	
enterVerificationCode(verificationCode: String): Response<PrepaymentInfo>	
requestPrepayment(itemCode: String, quantity: int): Response<String>	
getMyItems(): Response<List<Item>>	
updateStock(itemCode: String, quantity: int): Response<String>	
sendStockRequestMessage(itemCode:String, quantity:int): void	NetworkService
sendStockResponseMessage(dstId :String, itemCode:String): void	
sendPrepaymentInfoMessage(dstId :String, itemCode:String, quantity:int, verificationCode:String): void	
sendSaleRequestMessage(itemCode:String, quantity:int): void	
sendSaleResponseMessage(dstId :String, itemCode:String): void	
getSaleResponseMessages(itemCode: String): Message	
getStockResponseMessages(srcId :String, , itemCode:String, quantity:int): Message	
getMessages(messageType:MessageType): Vector<Message>	
changeWaitingMessageType(messageType:MessageType): void	
isEnough(itemCode:String, quantity:int):boolean	ItemService
getItemPrice(itemCode:String):int	
getItems():List<Item>	
getMyItems():List<Item>	
updateStock(itemCode:String, quantity:int):boolean	
getItemCount(itemCode:String):int	
count(itemCode:String):int	ItemRepository
findItem(itemCode:String):Item	
findAllItems():List<Item>	
findMyItems():List<Item>	
update(itemCode:String, quantity:int):void	
send(message:Message):void	Sender
run():void	
printDvmsNetworkInfo(ids:String[], ips:String[]):void	
changeWaitingMessageType(messageType:MessageType):void	Receiver
getResponseMessages():Vector<Message>	
clearResponseMessages():void	

CONTENTS

Design Patterns

Singleton → CardCompany, PrepaymentRepository, ItemRepository

기존:

각 클래스의 의존성 주입을 AppConfig에서 하며,
필요한 모든 객체를 생성해서 넘겨주는 방식이었습니다.
이 때 전체 시스템에서 하나의 객체를 사용해야 하는데,
이런 약속을 모르는 다른 개발자가 클래스를 직접 new해버리면 다른 객체가 생성되고 이를 사용하는 실수를 할 수 있습니다.

```
public class PrepaymentRepository {

    private final ConcurrentHashMap<String, PrepaymentInfo> savedPrepayments;

    public PrepaymentRepository() {
        savedPrepayments = new ConcurrentHashMap<>();
    }
}
```


Singleton 적용 후:

클래스 자체적으로 항상 하나의 Instance를 반환하도록 함으로써 이런 실수를 방지할 수 있습니다. 외부 생성자의 접근을 막기 위해 생성자를 private으로 했으며 getInstance 클래스를 static으로 만들었으며 이 때 바로 new하지 않고 Innerclass에서 생성해주는 방식을 사용하여 멀티쓰레드 환경에서도 안전하게 사용할 수 있습니다.

```
    private PrepaymentRepository() {}

    private static class PrepaymentRepositoryHelper{
        private static final PrepaymentRepository prepaymentRepository = new PrepaymentRepository();
    }

    public static PrepaymentRepository getInstance(){
        return PrepaymentRepositoryHelper.prepaymentRepository;
    }
```


Observer → ItemRepository, AdminPannel

기존:

자판기 화면과 관리자 화면을 이동할 때마다 수량을 바로 업데이트 하기 위해 AdminPanel 클래스를 새로 생성하였는데, 이 때 재고가 변경이 되지 않을 때에도 AdminPanel 클래스를 새로 생성하게 되는 문제가 있었습니다.

```
private void makeEvent() {
    adminBtn.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            super.mouseClicked(e);

            if (menu) {
                menu = false;
                cardPanel.remove(cardPanel.getComponent(1));
                adminPanel = new AdminPanel(controller);
                adminPanel.setName("admin");
                cardPanel.add("2", adminPanel);
            } else {
                menu = true;
            }
            cards.next(cardPanel);
        }
});
```

Observer 적용 후:

객체의 상태 변화를 관찰하는 옵저버들의 목록을 객체에 등록하여 상태변화가 있을 때마다 메서드 등을 통해 객체가 직접 옵저버에게 알리도록 하여 재고 변경이 있을 때만 이를 반영할 수 있게 했습니다.

```
private void makeEvent() {
      adminBtn.addMouseListener(new MouseAdapter() {
         @Override
         public void mouseClicked(MouseEvent e) {
              super.mouseClicked(e);
             cards.next(cardPanel);
      }
});
```

Design Patterns

Observer -> ItemRepository, AdminPannel

기존:

재고가 변경될 때 마다 매번 컨트롤러로부터 모든 아이템들의 재고를 직접 요청하고 화면에 적용하는 비효율적인 코드였습니다.

```
private void showStock() {
        typePanel = new JPanel[7];

        for (int i = 0; i < 7; i++) {
            typePanel[i] = new JPanel(new GridLayout(2, 1));

            JLabel item = new JLabel(myItems.get(i).getName());
            String itemCode = myItems.get(i).getItemCode();
            Response<Integer> stockResponse = controller.getItemCount(itemCode);
            JLabel num = new JLabel(stockResponse.getResult().toString());
            typePanel[i].add(item);
            typePanel[i].add(num);
            stockPanel.add(typePanel[i]);
        }
    }
```


Observer 적용 후:

재고가 변경되면 updateObserver 메소드가 자동으로 호출되고 변경이 일어난 아이템 코드와 수량이 넘어오기 때문에 해당하는 부분만 수정이 가능하게 되었습니다.

```
@Override
public void updateObserver(String itemCode, int quantity) {
    updateStockPanel(itemCode, quantity);
}
```

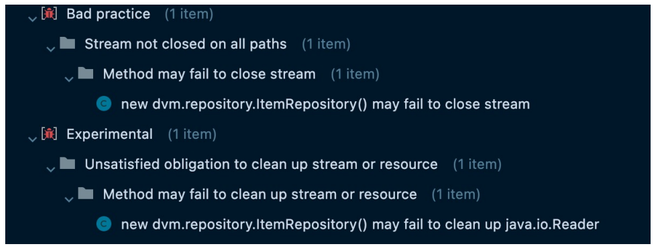
```
private void updateStockPanel(String itemCode, int quantity) {
    for (int i = 0; i < 7; i++) {
        if (itemCode.equals(myItems.get(i).getItemCode())) {
            JLabel number = (JLabel) typePanel[i].getComponent(1);
            number.setText(Integer.toString(quantity));
            System.out.println(number.getText());
            break;
        }
    }

}
```

SpotBugs

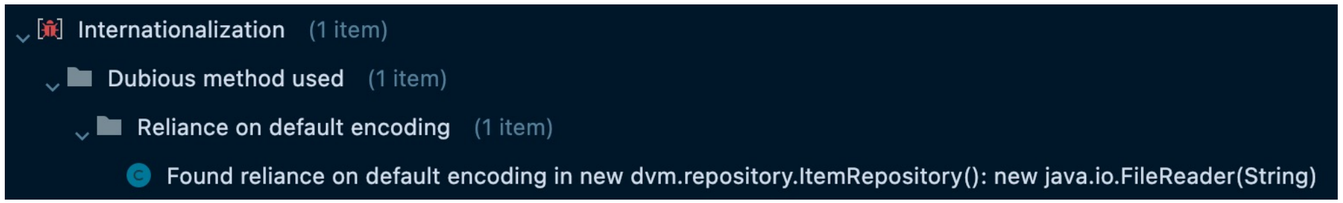
```
private ItemRepository() {
    try {
        Properties p = new Properties();
        p.load(new FileReader("path/stock.properties"));
    } catch (Exception e) {
        /* ... */
    }
}
```

Properties 파일을 읽어 재고 정보와 네트워크 정보를 설정하는 코드 [전]

Method may fail to close stream
Method may fail to clean up java.io.Reader
-> 파일 읽고 close 하지 않음 #1



파일 읽을 때 인코딩을 설정하지 않음 #2

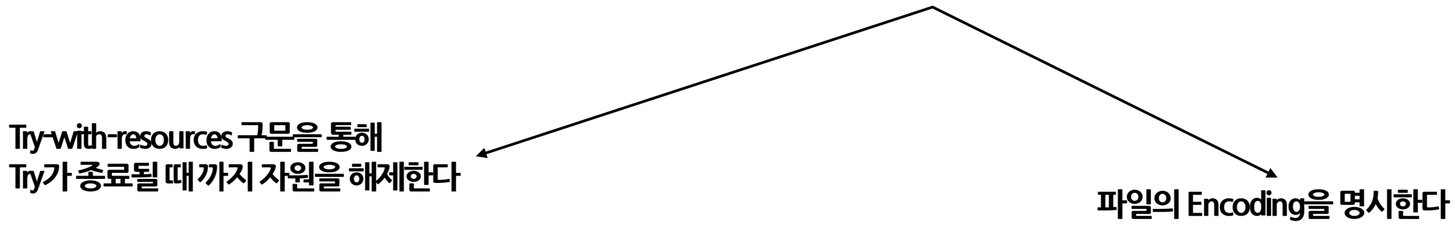


Clean Code

SpotBugs

```
private ItemRepository() {
   try (BufferedReader reader = new BufferedReader
           (new InputStreamReader(new FileInputStream("path/stock.properties"), StandardCharsets.UTF_8))) {
       Properties p = new Properties();
       p.load(reader);
   } catch (Exception e) {
       /* ... */
   }
}
```

Properties 파일을 읽어 재고 정보와 네트워크 정보를 설정하는 코드 [후]



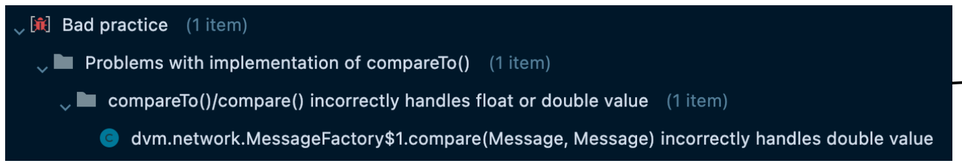
Clean Code

SpotBugs

```
messages.sort(new Comparator<Message>() {

    @Override
    public int compare(Message o1, Message o2) {
        double thisDistance = Math.sqrt(leftDifferX * leftDifferX + leftDifferY * leftDifferY);
        double otherDistance = Math.sqrt(rightDifferX * rightDifferX + rightDifferY *
rightDifferY);
        if (thisDistance == otherDistance) {
            if ((o1.getSrcId().compareTo(o2.getSrcId())) > 0) {
                return 1;
            }
        } else if (thisDistance > otherDistance) {
            return 1;
        }
        return -1;
    }

});
```

도착한 메세지를 거리순으로 정렬하는 코드 (전)

거리가 double형 변수인데 == 로 비교하여 문제 발생



Double.compare() 이용

```
messages.sort(new Comparator<Message>() {

    @Override
    public int compare(Message o1, Message o2) {
        double thisDistance = Math.sqrt(leftDifferX * leftDifferX + leftDifferY * leftDifferY);
        double otherDistance = Math.sqrt(rightDifferX * rightDifferX + rightDifferY *
rightDifferY);
        if (Double.compare(thisDistance, otherDistance) == 0) {
            if ((o1.getSrcId().compareTo(o2.getSrcId())) > 0) {
                return 1;
            }
        } else if (Double.compare(thisDistance, otherDistance) == 1) {
            return 1;
        }
        return -1;
    }

});
```

도착한 메세지를 거리순으로 정렬하는 코드 (후)

Return null, 0, -1 대신 optional 이용

```java
public int getItemPrice(String itemCode) {
        Item item = itemRepository.findItem(itemCode);
        if (item == null) {
            return -1;
        }
        return item.getPrice();
    }
```



```java
public int getItemPrice(String itemCode) throws IllegalArgumentException {
    return itemRepository.findItem(itemCode)
            .orElseThrow(() -> new IllegalArgumentException("wrong item code"))
            .getPrice();
}
```

Opinions

개발하면서 느낀 점 객관적이면서 주관적으로

Cycle을 여러 번 돌면서 진행 했어야 했는데 두번씩 밖에 못 돈 거 같아서 아쉽다.
Diagram부터 차근차근 만들어가는 것이 처음인데, 스켈레톤 코드를 이용한 개발 경험이 신기했다.
OOPT를 통해서 여러 다이어그램을 그리는 방법을 알게 되었고, 다른 수업에서도 설계서를 작성할 때 많은 도움이 되었다.
지금까지 해본 프로젝트와 달리 코드 구현에 앞서 분석, 설계라는 사전 단계를 거쳤다. 코드 없이 설계하면서 어려운 점도 많았지만 이전보다 확실히 군더더기 없는 코드가 나왔음을 알 수 있었다. 설계 과정에서 객체간의 협력, 메시지 패싱등을 고려하면서 객체지향적으로 사고하는것이 어느정도 강제되었는데 이런점이 OOAD의 큰 장점중 하나인것 같다.

OOPT Final Presentation

객체지향개발방법론

Object Oriented Programming

감사합니다