Distributed Vending Machine

Team 5 / 2nd Demonstration

5조

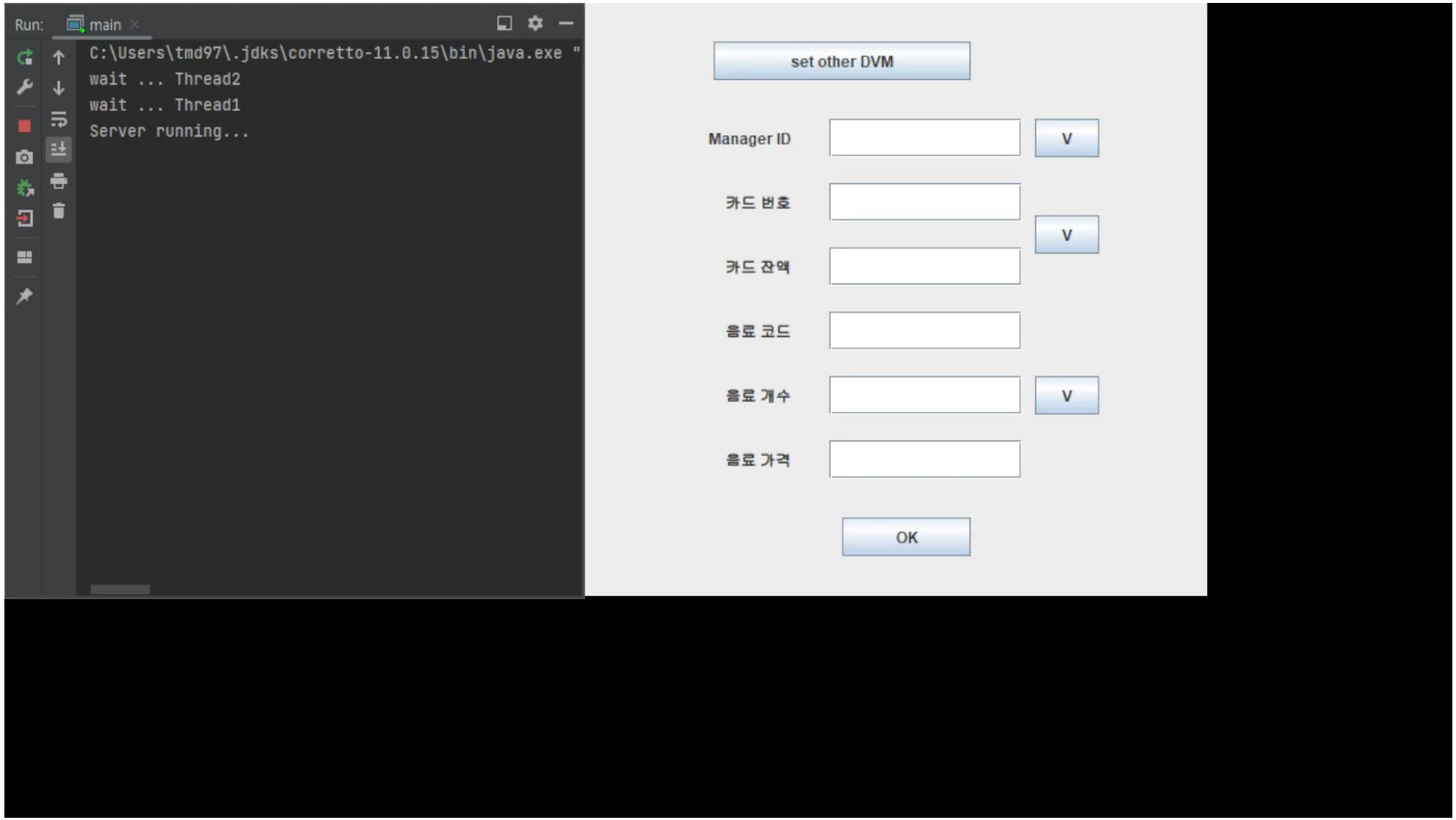
201711310 변건우

202011313 손윤환

202014186 오상희

201912698 이현규

01 Demonstration



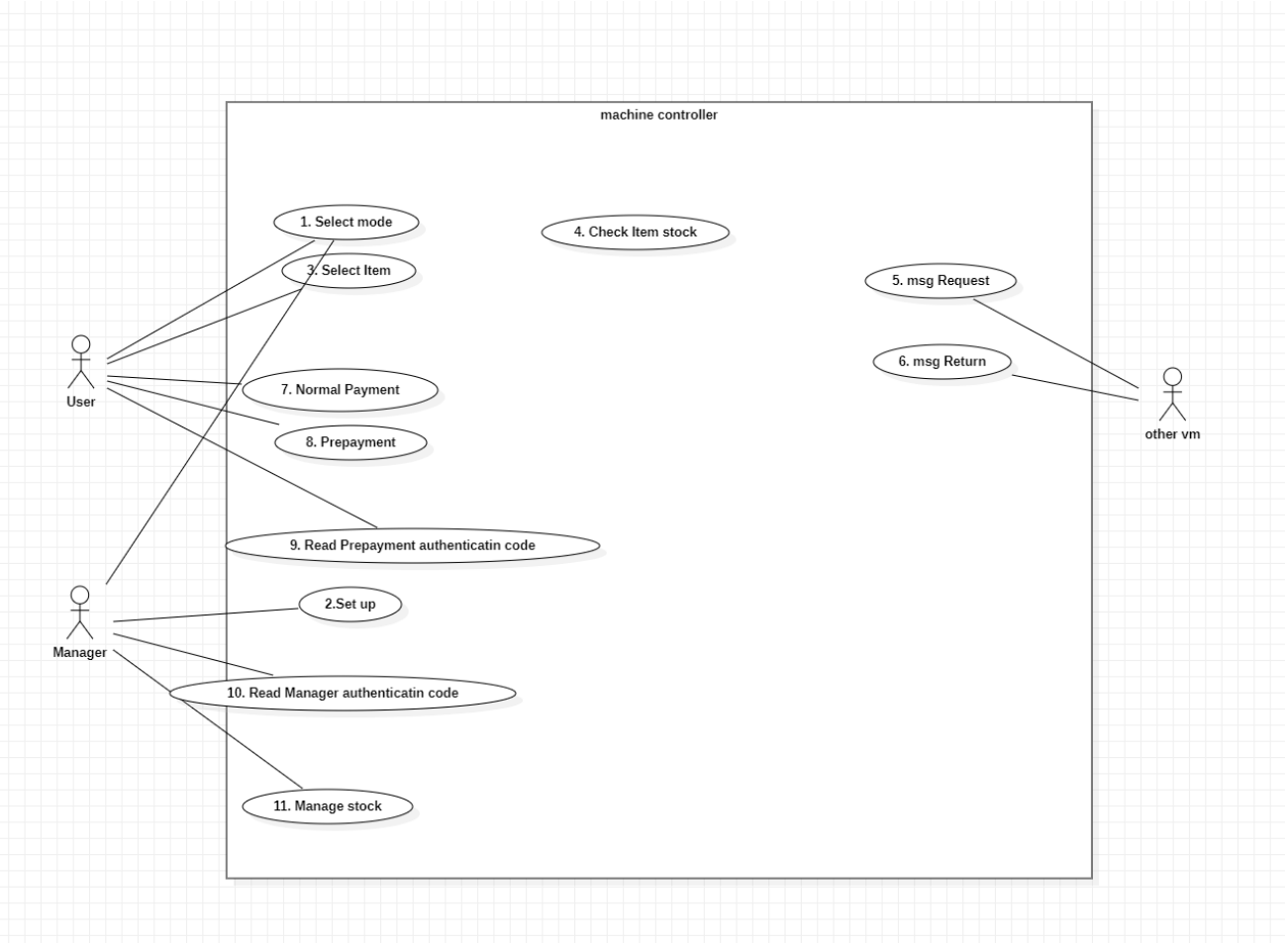
02 Define Business Use Case & Diagram

User : 자판기를 사용하기 위해 시스템과 상호작용하는 actor

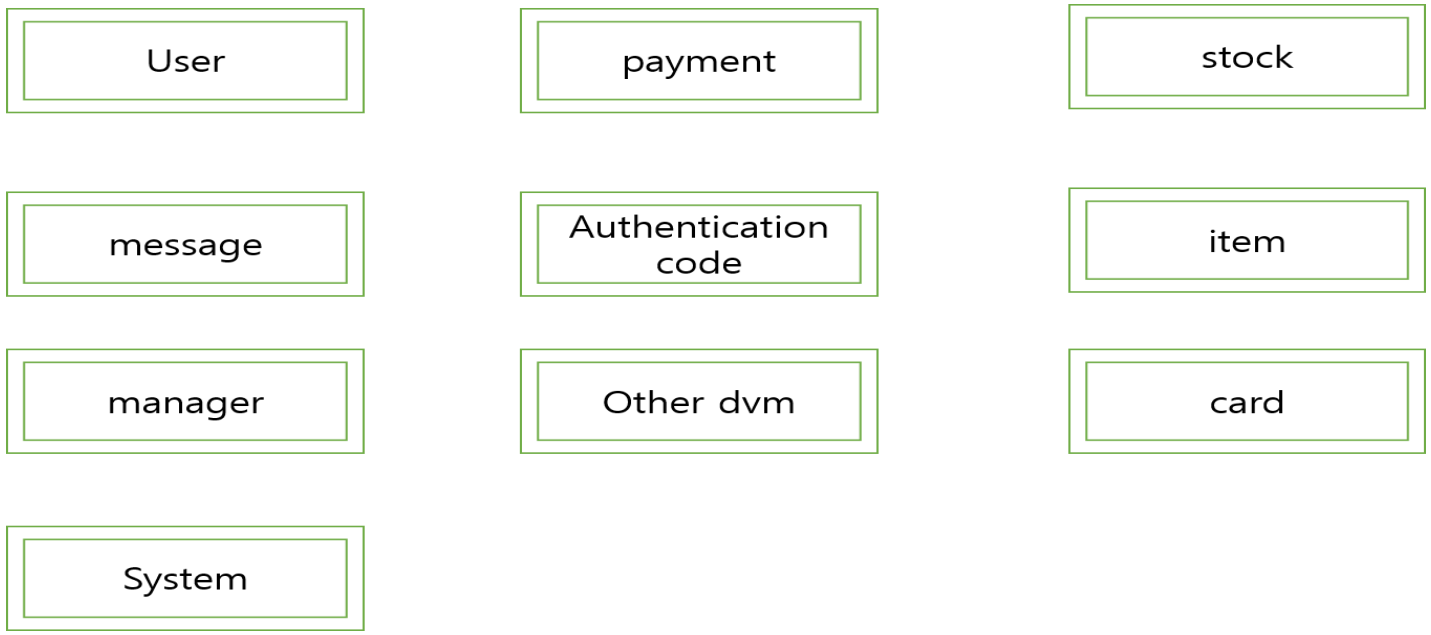
Manager : 자판기 재고를 관리하기 위해 시스템과 상호작용하는 actor

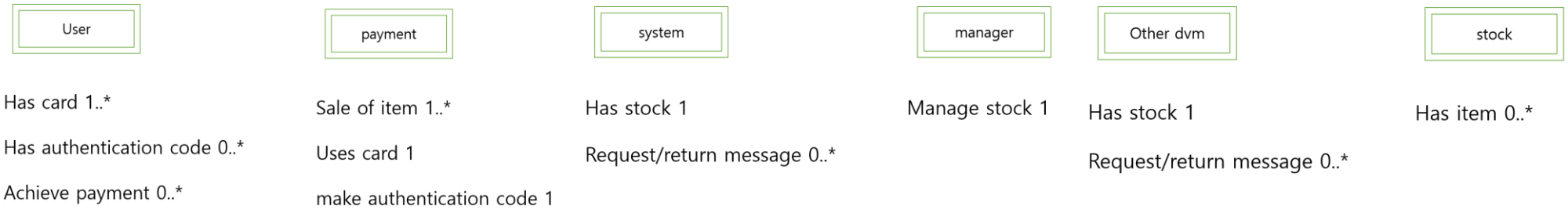
Other VM : 시스템과 상호작용 하는 다른 자판기 actor

Actor	Use cases
User	Select mode
	Select item
	Normal payment
	Prepayment
	Read Prepayment authentication code
Manager	Set up
	Read Manager authentication code
	Manage stock
Other VM	msg Request
	msg Return

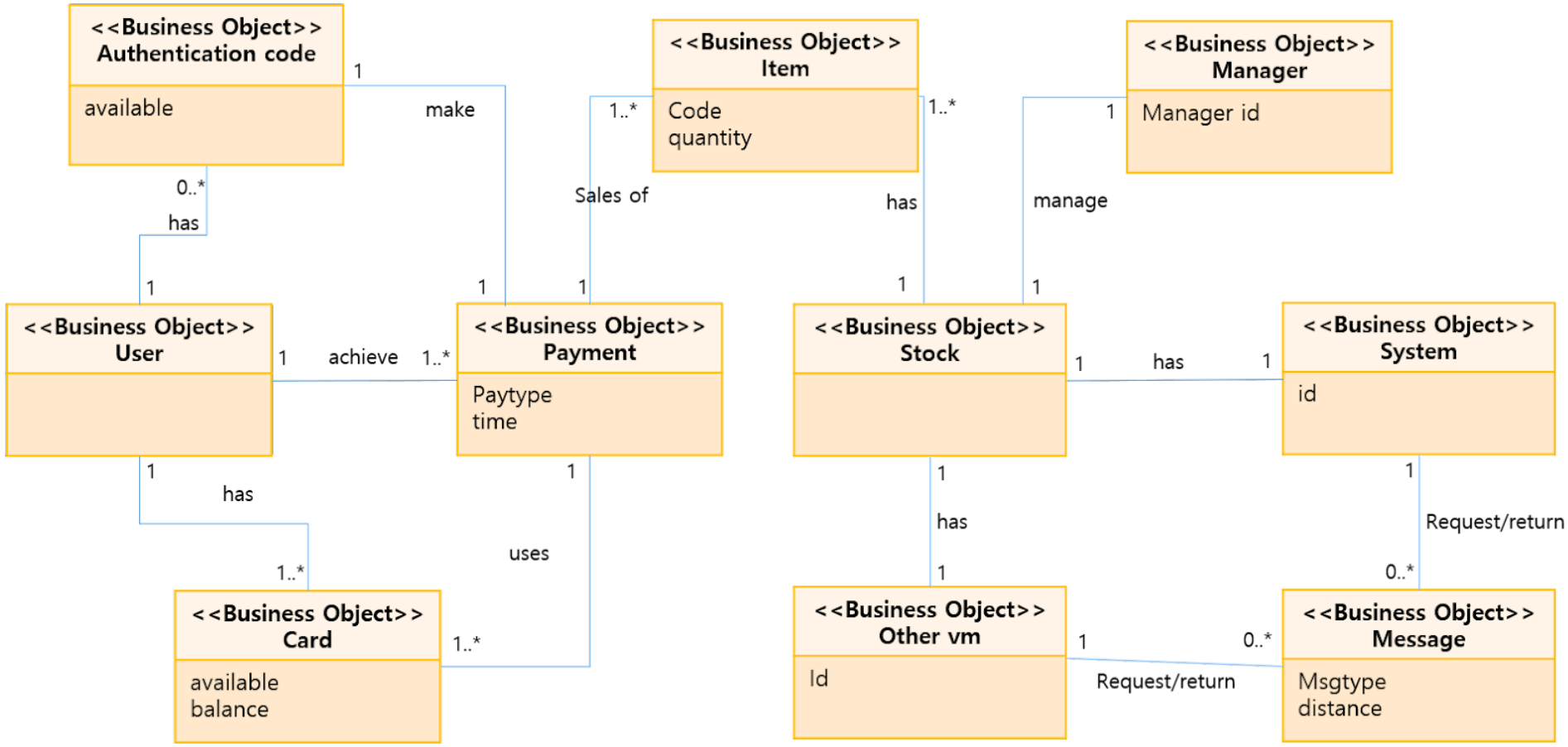


03 Define Domain Model



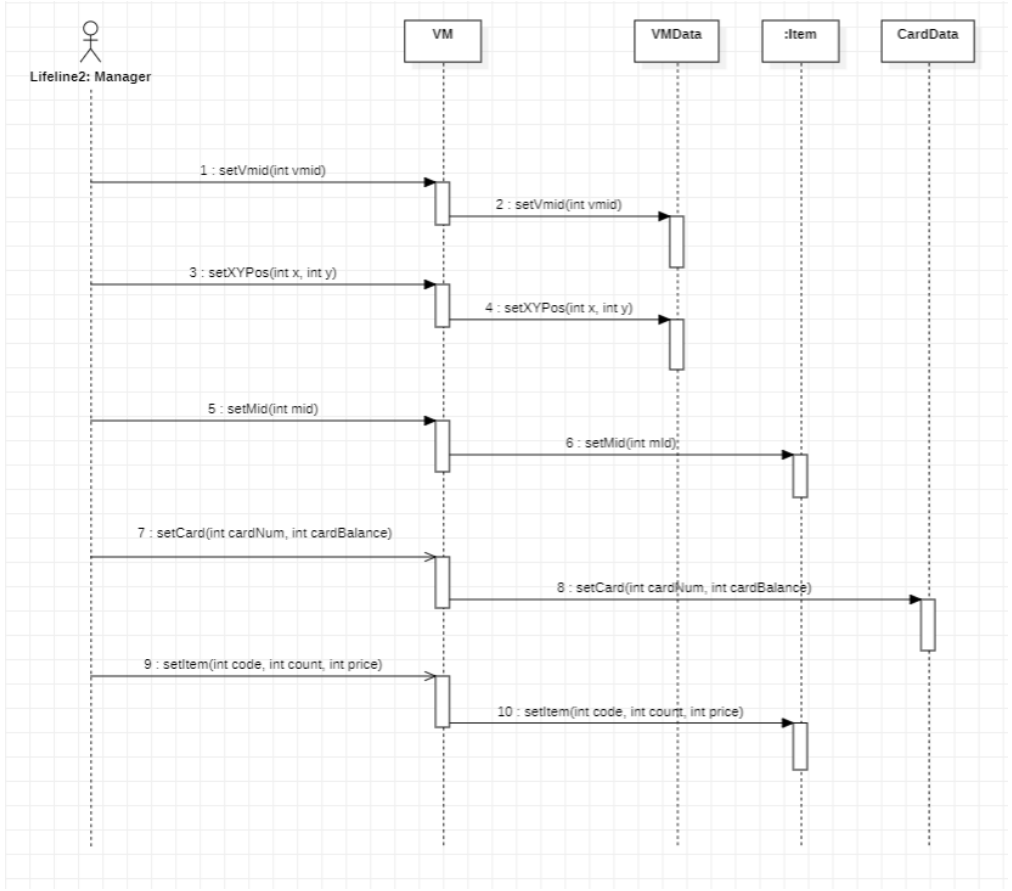


03 Define Domain Model

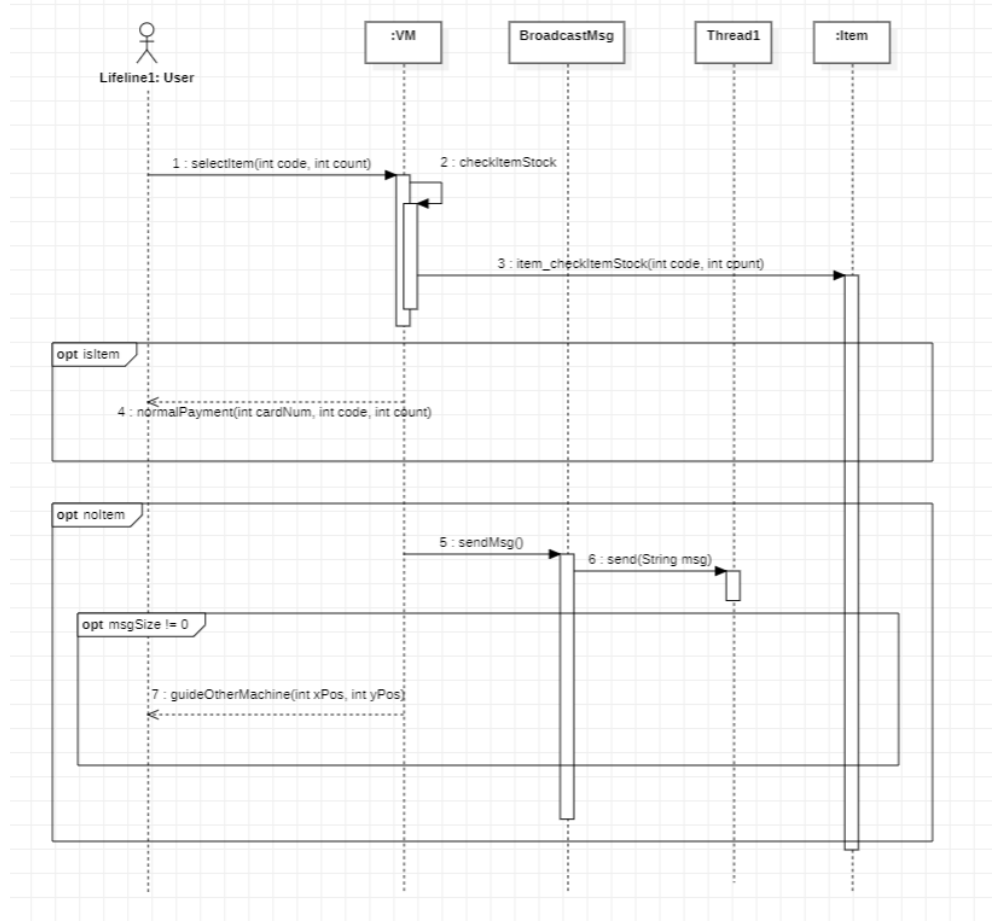


04 Define System Sequence Diagrams

01. SetUp

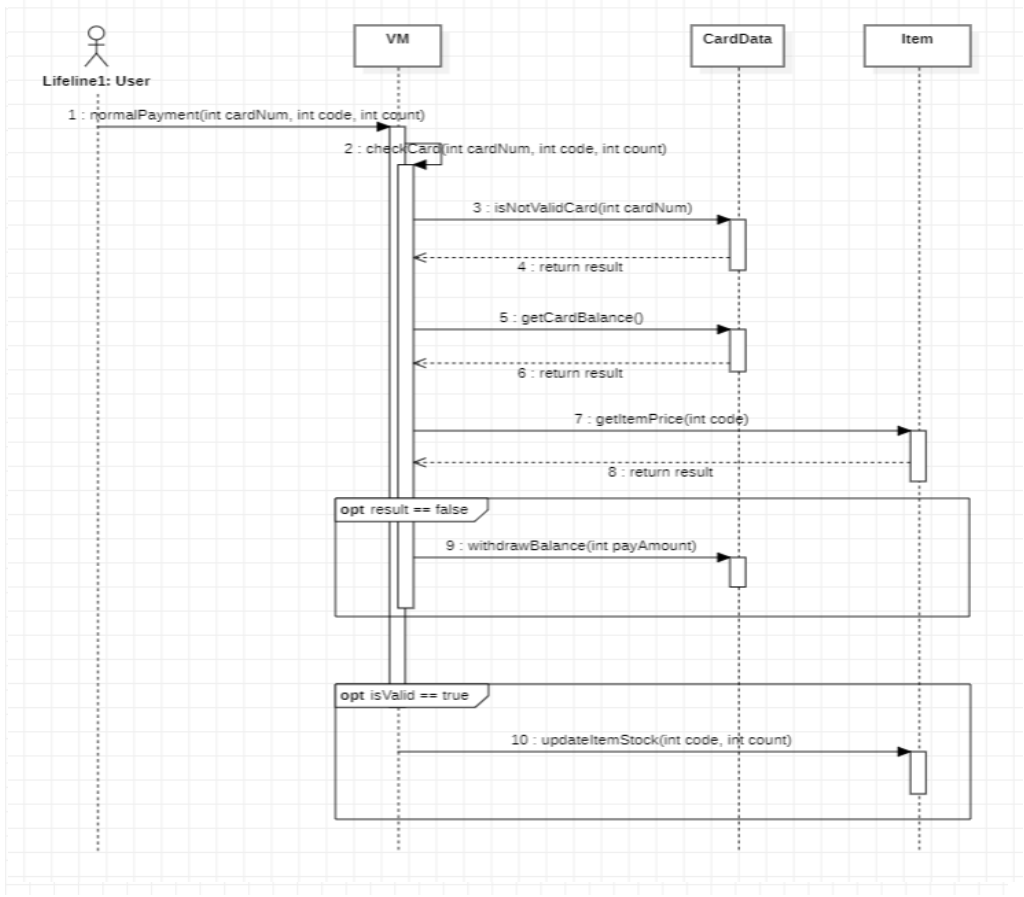


02. Select Item

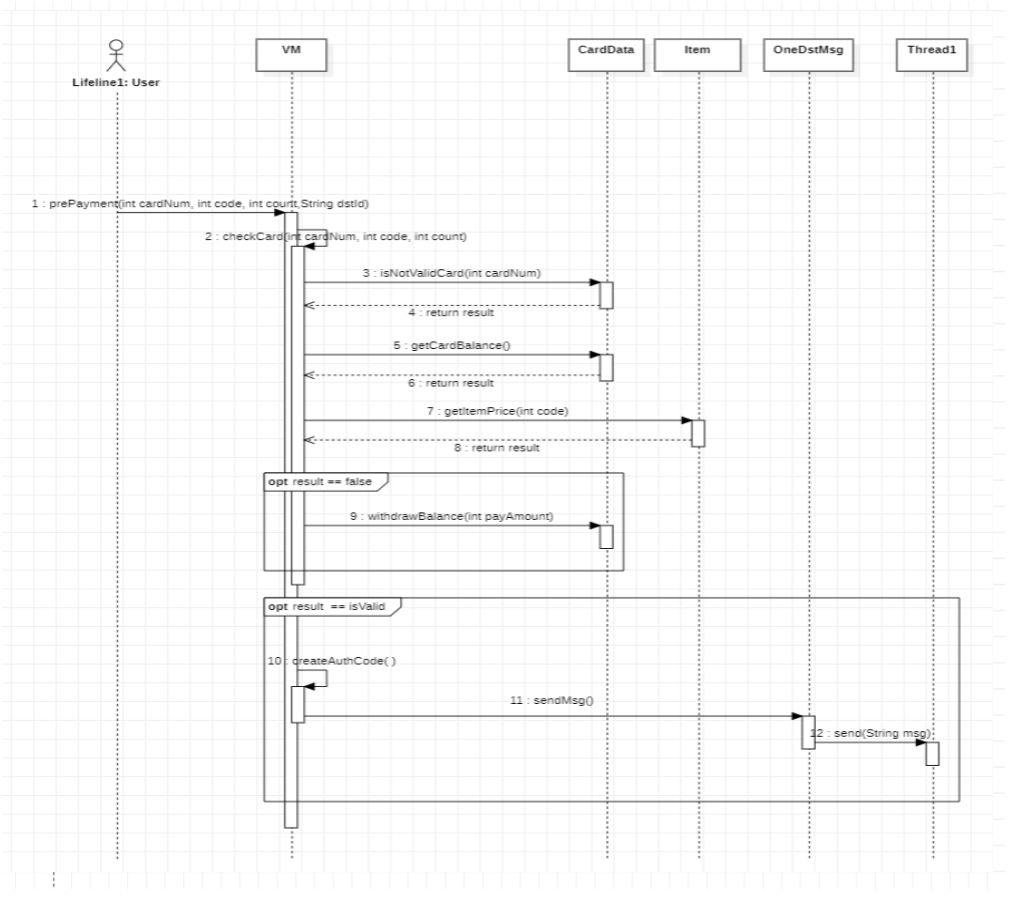


04 Define System Sequence Diagrams

03. NormalPayment

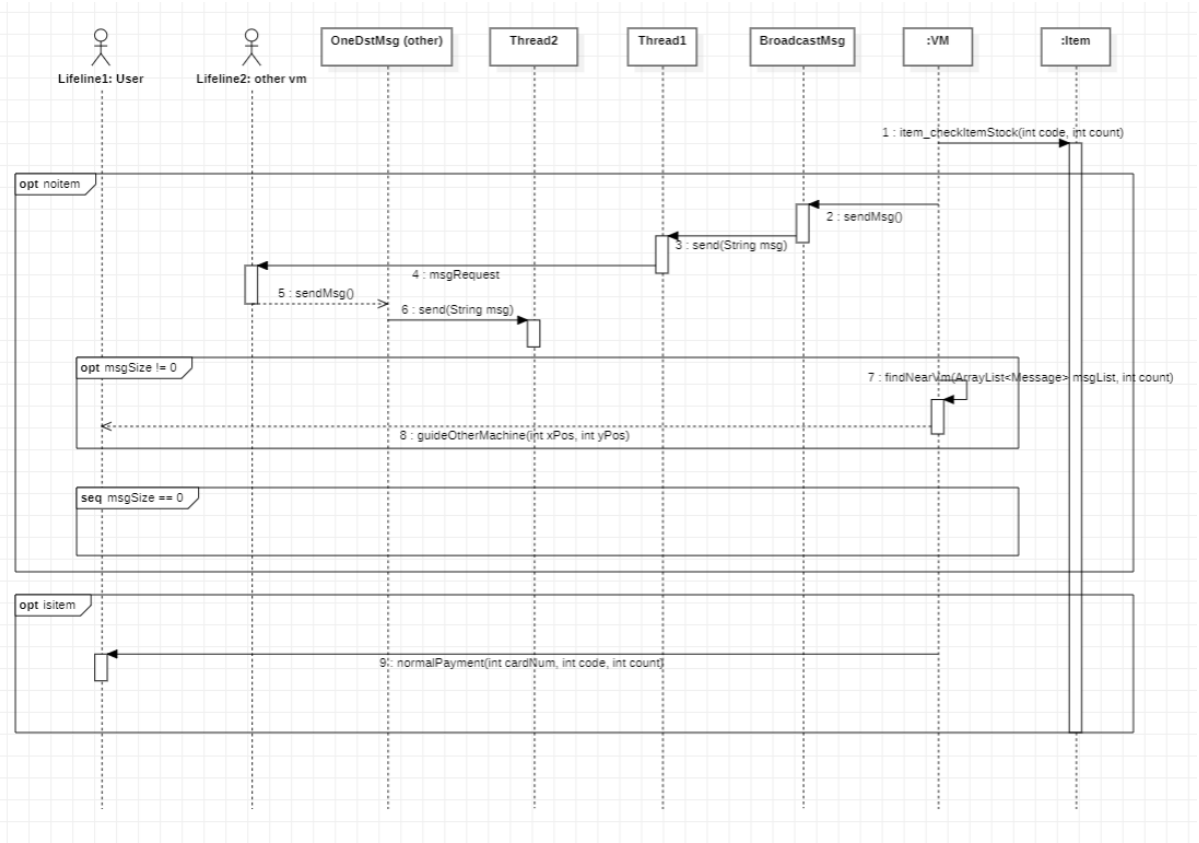


04. PrePayment

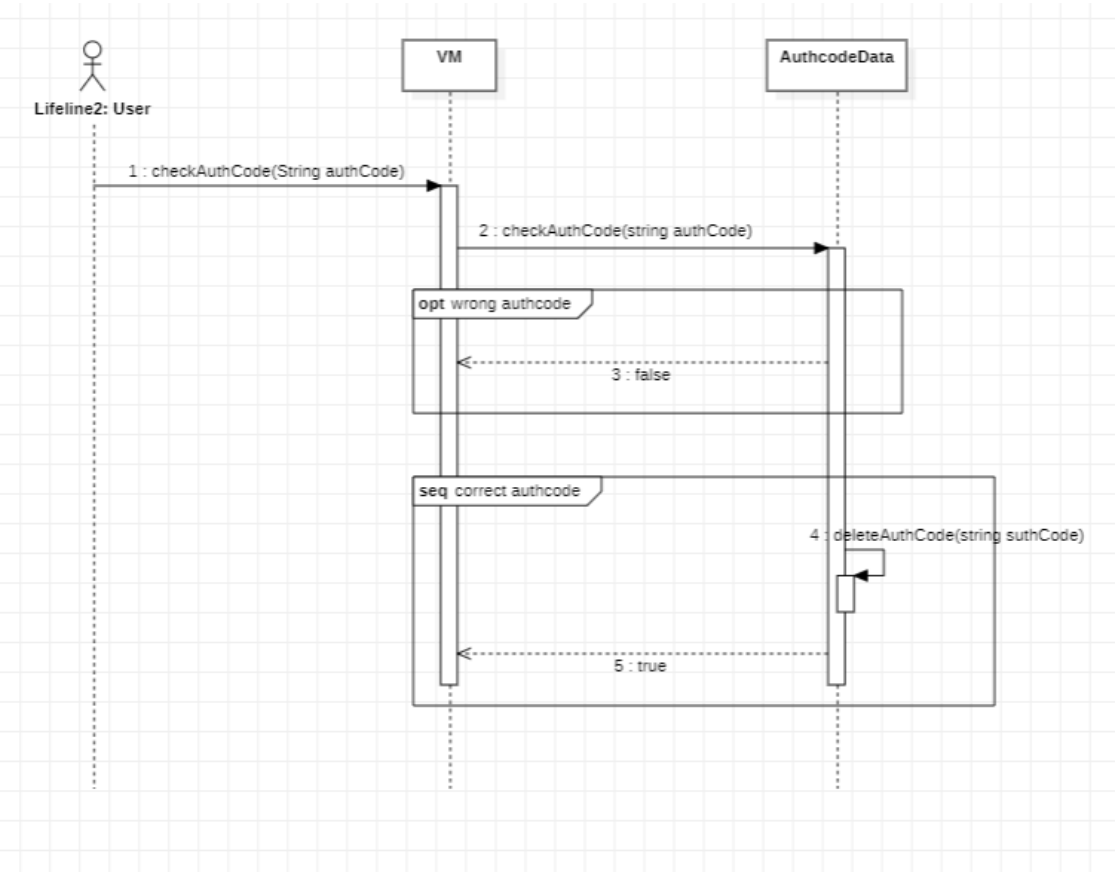


04 Define System Sequence Diagrams

05. Check Item Stock

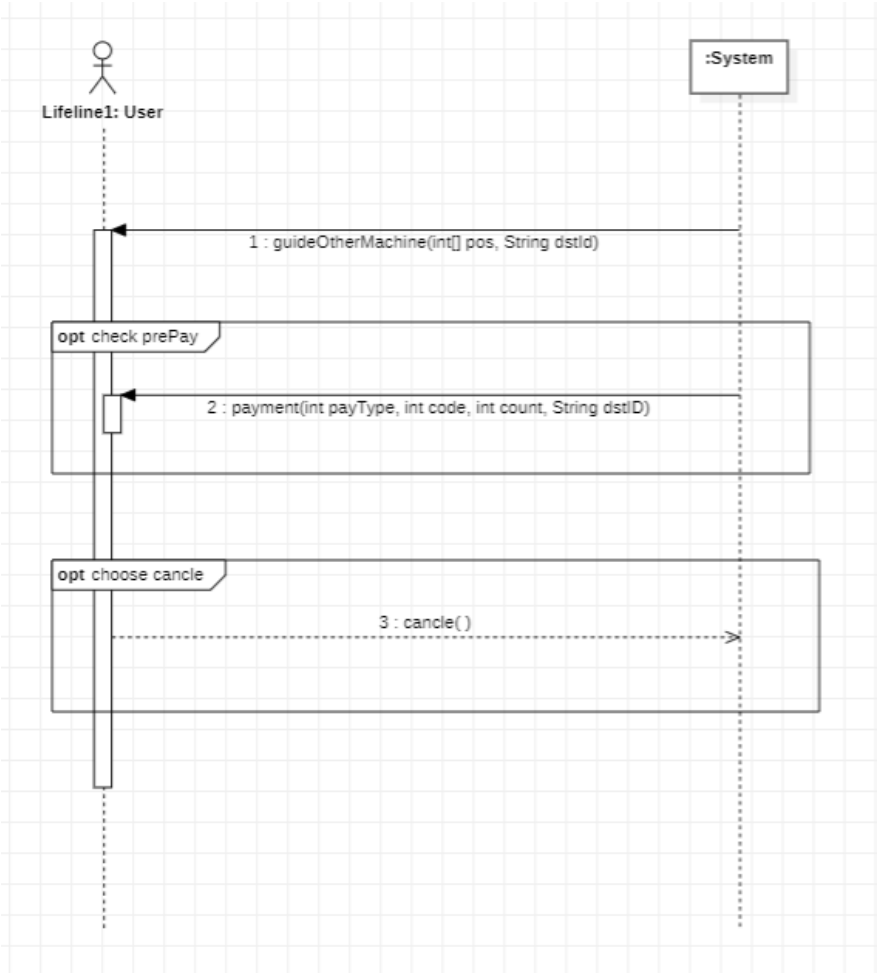


06. Read prepayment authentication code

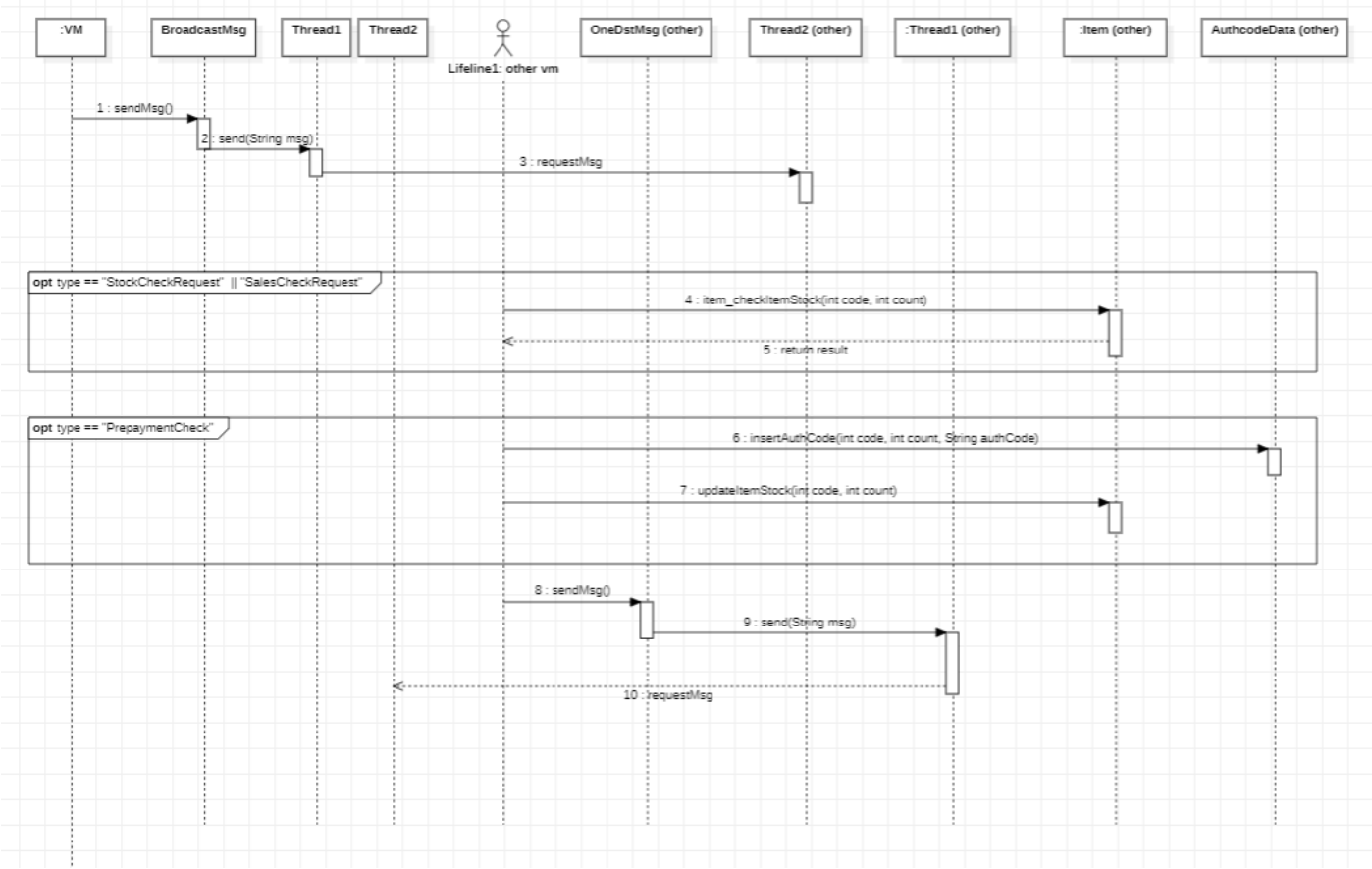


04 Define System Sequence Diagrams

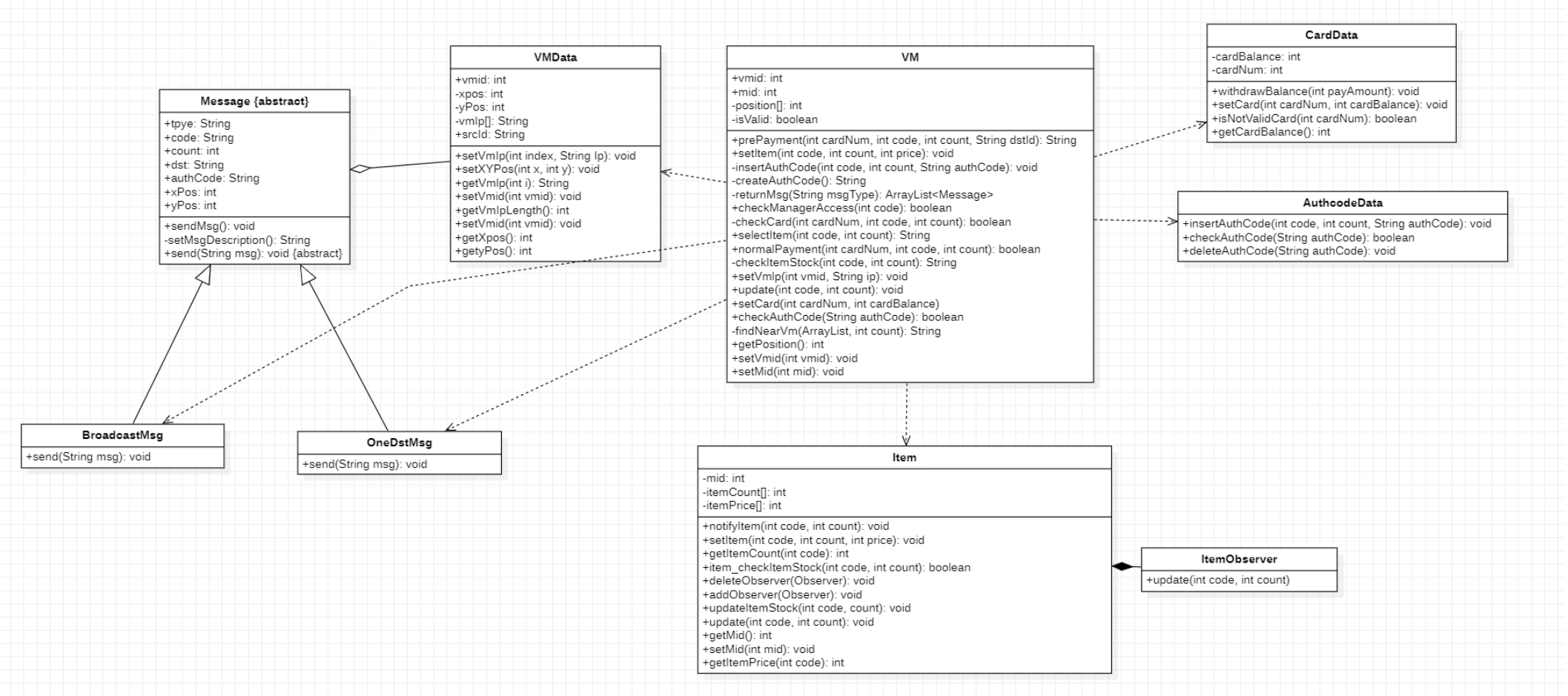
07. Guide other machine



08. Message request and return



05 Design Class Diagram



06 System Testing

Num	Ref. #	Use Case	Title	Description	Pass
1	R 1.1	Select mode	메뉴 선택 시험	자판기 시스템의 3가지 모드 실행 test 1. 음료수 선택 시 음료수 선택 화면 이동 2. 선결제 선택 시 선결제 코드 입력 화면 이동 3. 매니저 선택 시 매니저 코드 입력 화면 이동	O
2	R 1.2	Set up	자판기 설정 시험	1. 자판기 초기 설정 시 각 설정 값이 제대로 설정되는지 확인 2. IP, 좌표 설정 시 각 설정 값이 제대로 설정되는지 확인	O
3	R 1.3	Select item	음료 선택 시험	음료선택 화면에서 사용자가 선택한 음료코드와 개수만큼 주문되는지 확인	O
4	R 1.3	Select item	음료 선택 예외 처리 시험	음료코드나 개수를 입력하지 않고 다음 단계가 진행이 안되는 지 확인	O
5	R 4.2	Check item stock	재고 확인 시험	음료 주문 시 해당 음료의 재고와 코드를 입력 받은 대로 확인하는 지 확인	O
6	R 2.1	msg Request	메시지 요청 시험	메시지를 보내는 4가지 경우(StockCheckRequest, StockCheckResponse, PrePaymentCheck, SalesCheckResponse)에 대한 전송 여부 확인	O
7	R 2.1	msg Request	재고 확인 응답 처리 시험	StockCheckResPonse 응답 메시지를 받은 후 findNearVM 진행되는지 확인	O
8	R 2.2	msg Return	재고 확인 응답 시험	SaleCheckRequest 요청을 받은 후 checkItemStock과 StockCheckResponse가 순서대로 진행되는지 확인	O
9	R 2.3	Find near VM	최단거리 자판기 탐색 시험	입력 받은 자판기들 중 최단거리의 자판기 선택 후 반환 및 안내 확인	O

06 System Testing

Num	Ref. #	Use Case	Title	Description	Pass
10	R 3.1	Normal payment	일반 결제 시험	재고가 존재할 경우 진행되는지 확인	O
11	R 3.1	Normal payment	일반 결제 예외 처리 시험	결제 도중 취소 요청이 가능하지 확인	O
12	R 3.2	Prepayment	선결제 시험	Find near Vm 실행 후 선결제가 진행되는지 확인	O
13	R 3.2	Prepayment	선결제 예외 처리 시험	1. 결제 중 재고가 부족해 취소되는지 확인 2. 결제 중 사용자의 취소 요청 가능한지 확인	O
14	R 3.3	Get card	카드 요청 시험	카드 요청 화면이 출력되는지 확인	O
15	R 3.4	Check card validation	카드 유효성 검사 시험	카드가 유효하지 않을 경우 재 요청하는지 확인	O
16	R 3.5	Check card balance	카드 결제 시험	1. 결제 중 카드 잔고 확인하는지 확인 2. 결제 후 성공 화면 출력 확인	O
17	R 3.6	Create authentication code	인증 코드 생성 시험	생성한 문자열이 10자리 영문자+숫자인지 확인	O
18	R 3.7	Show authentication Code	인증 코드 출력 시험	화면을 통해 인증코드를 출력하는 지 확인	O
19	R 3.9	Check authentication code	인증 코드 확인 시험	사용자가 인증코드 입력 시 인증코드 존재 여부 및 반환 결과 확인	O
20	R 3.10	Delete authentication code	인증 코드 삭제 시험	사용된 인증코드가 삭제되는 지 확인	O
21	R 4.1	Update item stock	자판기 재고 최신화 시험	음료수 구매 이후 자판기 재고 업데이트 여부 확인	O
22	R 5.1	Manager Access	관리자 모드 전환 시험	1. 인증코드 불일치 시 재 요청 하는지 확인 2. 인증 후 관리자 모드에 들어가는지 확인	O

07 Design Pattern – Template Method

Before

```
private void requestMsg(String type, String code, int count, String dst, String authCode, int xPos, int yPos) throws InterruptedException {
    Message msg = new Message();
    Message.MessageDescription msgDesc = new Message.MessageDescription();
    Serializer msg2json = new Serializer();


    code = String.format( "%1$02d" , Integer.parseInt(code));
    System.out.println("바꾼 code: "+code);

    msg.setSrcId(vmData.srcId); //우리 Id는 5로 고정 => 얘도 item에 저장하고 가져오는 식으로 해야하나...? (귀찮음 ㅎㅎ)
    msg.setDstID(dst);

    msg.setMsgType(type);
    msgDesc.setItemCode(code);
    msgDesc.setItemNum(count);
    msgDesc.setDvmXCoord(xPos); //이것도 위와 같은 의견. setup에서 정의하지 않음. 아마 연결하기 전에 값 줄건데 그걸로 고정
    msgDesc.setDvmYCoord(yPos); //이하동문.
    msgDesc.setAuthCode(authCode);
    msg.setMsgDescription(msgDesc);

    String jsonMsg = msg2json.message2Json(msg); //msg=>json

    if(dst.equals("0")){
        for(int i=0;i< vmData.getVmIpLength();i++){
            if(vmData.getVmIp(i).equals("our")||vmData.getVmIp(i).equals("null")){
                continue;
            }

            DVMClient client = new DVMClient(vmData.getVmIp(i), jsonMsg);
            client.run();
        }
    }
    else{

        String tmpDst = Character.toString(dst.charAt(dst.length()-1));

        System.out.println("tmpDst: "+tmpDst);
        DVMClient client = new DVMClient(vmData.getVmIp( i: Integer.parseInt(tmpDst)-1), jsonMsg);
        client.run();
    }
}
```

메시지를 보내는 부분이
전부 하나로 합쳐져 있어 호출시마다
어떤 메시지를 보내는 것인지 헷갈린다.

07 Design Pattern – Template Method

After 1

```
public class BroadcastMsg extends Message {

    BroadcastMsg(VMData vmData, String type, String code, int count, String dst, String authCode, int xPos, int yPos) {
        super(vmData, type, code, count, dst, authCode, xPos, yPos);
    }

    @Override
    void send(String msg) {
        for(int i=0;i< vmData.getVmIpLength();i++){
            if(vmData.getVmIp(i).equals("our")||vmData.getVmIp(i).equals("null")){
                continue;
            }

            DVMClient client = new DVMClient(vmData.getVmIp(i), msg);
            try {
                client.run();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

메시지의 종류에 따라 구체화하는
하위 클래스를 만들었다.

```
//requestMsg("StockCheckRequest",String.format( "%1$02d" , code),count,"0","",vmData.getXpos(),vmData.getyPos());
new broadcastMsg(vmData, type: "StockCheckRequest",String.format( "%1$02d" , code),count, dst: "0", authCode: "",vmData.getXpos(),vmData.getyPos());
```

07 Design Pattern – Template Method

After 2

```java
public class OneDstMsg extends Message {
    OneDstMsg(VMData vmData, String type, String code, int count, String dst, String authCode, int xPos, int yPos) {
        super(vmData, type, code, count, dst, authCode, xPos, yPos);
    }

    @Override
    void send(String msg) {
        String tmpDst = Character.toString(dst.charAt(dst.length()-1));

        System.out.println("tmpDst: "+tmpDst);
        DVMClient client = new DVMClient(vmData.getVmIp( i: Integer.parseInt(tmpDst)-1), msg);
        try {
            client.run();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

장점:
메시지의 종류에 따라
Message 클래스를 상속받아 사용하여
코드 내 가독성이 높아졌다.

새로운 메세지들의 추가가 용이해졌다.

단점:
클래스 수가 증가해 클래스 관리가 복잡해졌다.

```java
//requestMsg("PrepaymentCheck" ,Integer.toString(code), count,dstId,authCode,vmData.getXpos(),vmData.getyPos()); //dst
new onedstMsg(vmData,  type: "PrepaymentCheck" ,Integer.toString(code), count,dstId,authCode,vmData.getXpos(),vmData.getyPos());
```

07 Design Pattern – Observer

Before

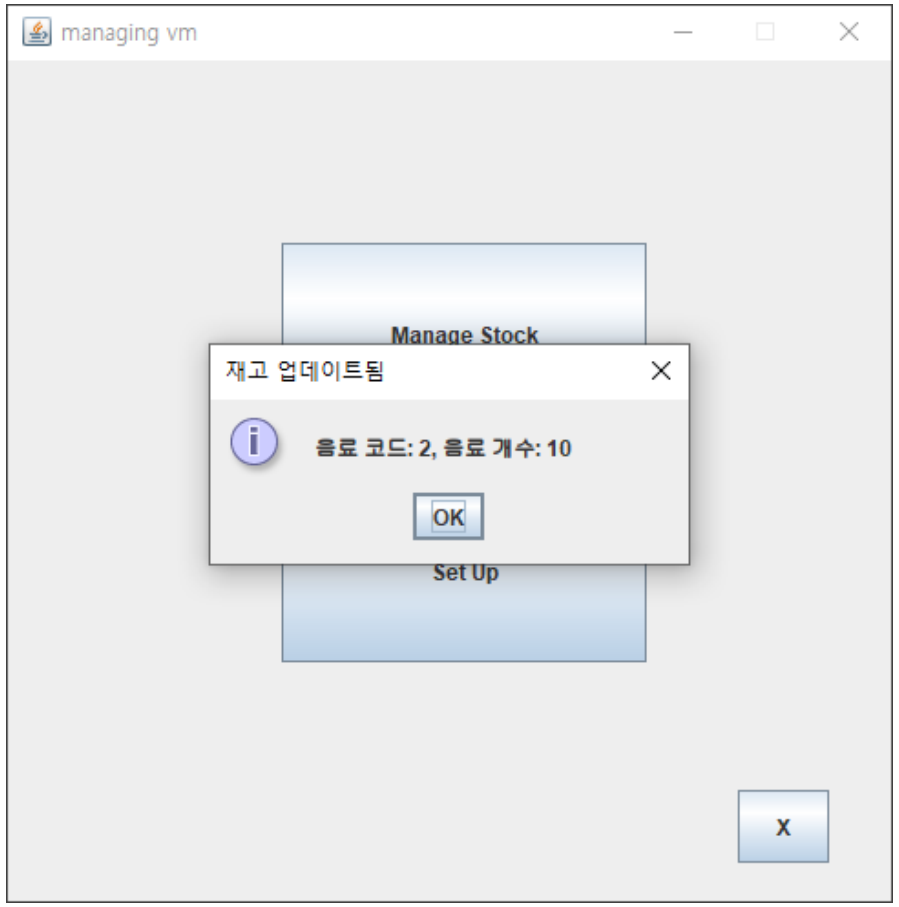
```
@Override
public void addObserver(Observer observer) { observers.add((ItemObserver) observer); }

@Override
public void deleteObserver(Observer observer) { observers.remove((ItemObserver) observer); }

@Override
public void notifyItem(int code, int count) {
    for(ItemObserver o : observers){
        o.update(code, count);
    }
}
```

매니저 모드에 진입 시
Other VM에 대한 선결제가 완료된 경우,
매니저가 재고에 대한 정보를
알기가 어려워 옵저버 패턴을 적용했다.

```
public class ItemObserver implements Observer{
    @Override
    public void update(int code, int count) {
        JOptionPane jOptionPane = new JOptionPane();
        jOptionPane.showMessageDialog( parentComponent: null,  message: "음료 코드: " + code + ", 음료 개수: " + count,  title: "재고 업데이트됨", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

07 Design Pattern – Observer

After 1

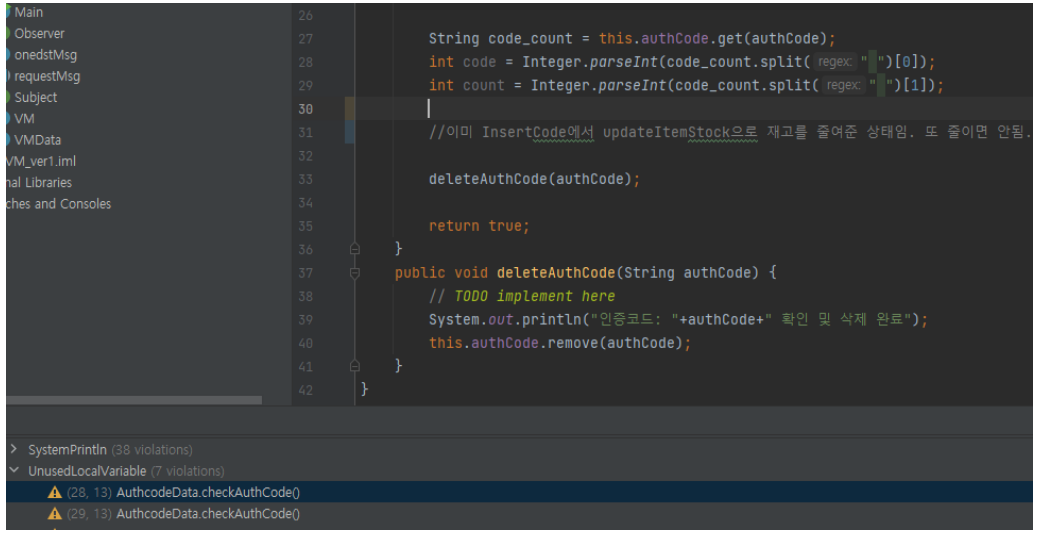


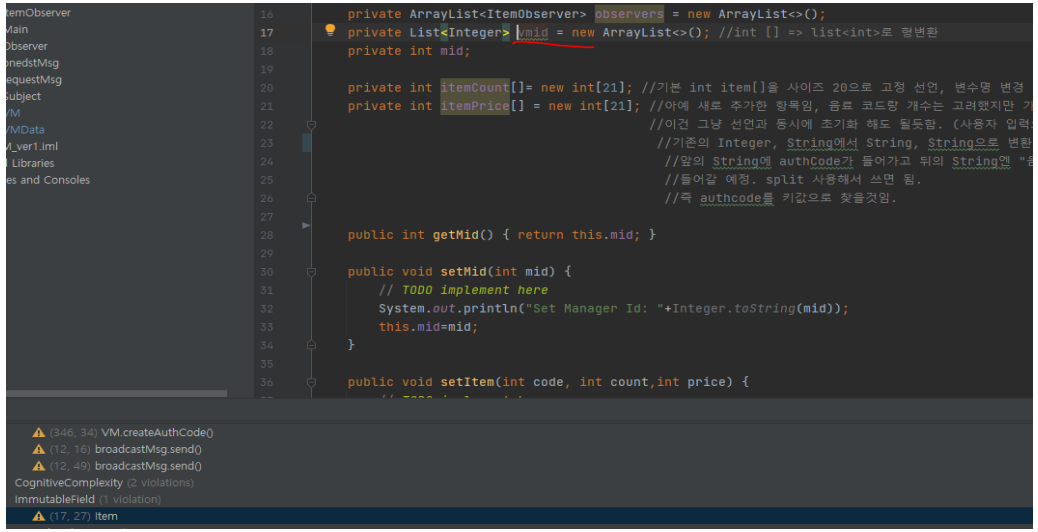
장점:
재고 변경에 대한 알림을 통해 매니저의 재고 관리가 용이하다.

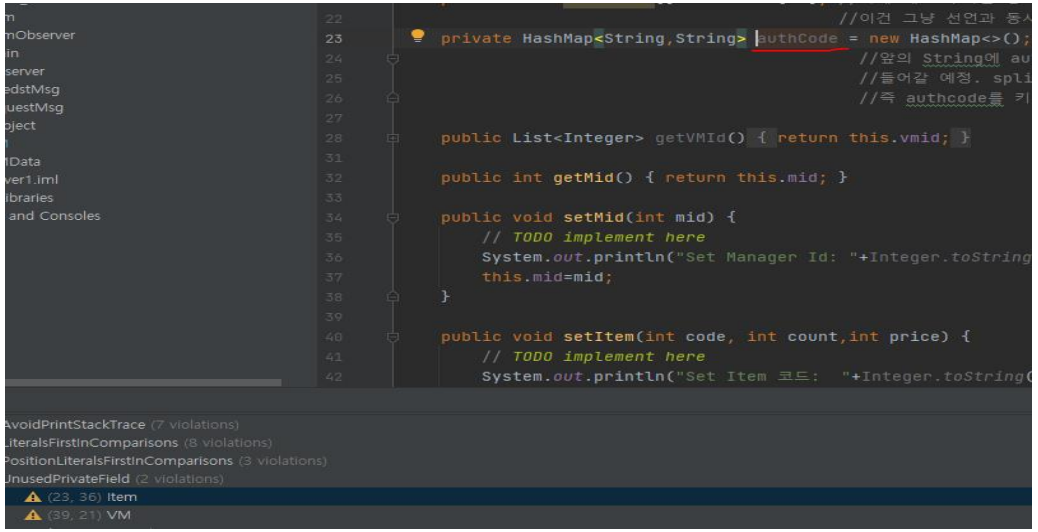
다양한 옵저버 객체를 생성하여 다른 데이터에 대한 감시 또한 가능하다.

단점:
옵저버가 많아질수록 프로그램 내에서 객체 간의 접근, 연산이 증가하여 전체 속도가 느려질 수 있다.

08 Clean Code - 미사용 변수 삭제







08 Clean Code – 변수명 변경

Before

```
JButton selectMode_item_btn = new JButton( text: "음료 선택");
JButton selectMode_prepay_btn = new JButton( text: "선결제 수령");
JButton selectMode_manager_btn = new JButton( text: "매니저");

selectMode_item_btn.addActionListener(new ActionListener() { //음료
    @Override
    public void actionPerformed(ActionEvent e) {
        selectItem();
        frame.dispose();
    }
});

selectMode_prepay_btn.addActionListener(new ActionListener() { //선
    @Override
    public void actionPerformed(ActionEvent e) {
        readPrepayAuthCode();
        frame.dispose();
    }
});

selectMode_manager_btn.addActionListener(new ActionListener() { //
    @Override
```

After

```
JButton btnItem = new JButton( text: "음료 선택");
JButton btnPrepay = new JButton( text: "선결제 수령");
JButton btnManager = new JButton( text: "매니저");

btnItem.addActionListener(new ActionListener() { //음료 선택
    @Override
    public void actionPerformed(ActionEvent e) {
        selectItem();
        frame.dispose();
    }
});

btnPrepay.addActionListener(new ActionListener() { //선결제 수령
    @Override
    public void actionPerformed(ActionEvent e) {
        readPrepayAuthCode();
        frame.dispose();
    }
});
```

08 Clean Code – 변수명 변경

Before

```
public boolean checkItemStock(int code, int count) {
    // TODO implement here
    System.out.print("code: "+Integer.toString(code)+" cou
    if(itemCount[code]>=count){ //재고 확인해서 주문수량 보다 ㅋ
        System.out.print("있음");
        System.out.println();
        return true;
    }
    System.out.print("없음");
    System.out.println();
    return false;
}
```

After

```
public boolean item_checkItemStock(int code, int count) {
    // TODO implement here
    System.out.print("code: "+Integer.toString(code)+" count:"
    if(itemCount[code]>=count){ //재고 확인해서 주문수량 보다 크거나
        System.out.print("있음");
        System.out.println();
        return true;
    }
    System.out.print("없음");
    System.out.println();
    return false;
}
private String checkItemStock(int code, int count) throws InterruptedException { //
    // TODO implement here
```

08 Clean Code – 함수 효율화

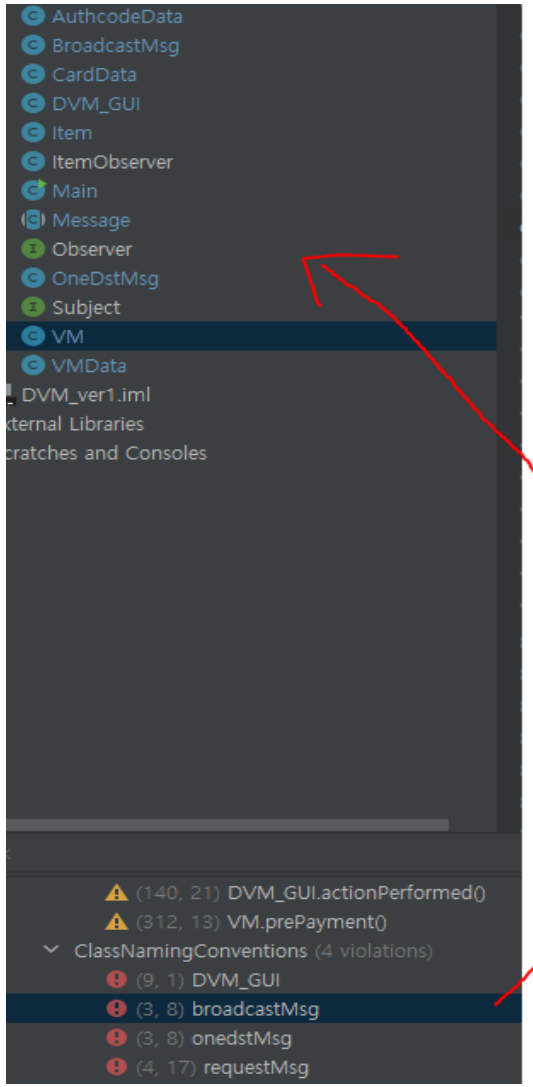
Before

```java
public boolean isNotValidCard(int cardNum){
    // 이건 isValidCard로 해도 상관없는데 저 편한대로 했어요
    if(this.cardNum != cardNum){
        // System.out.println("카드 번호가 유효하지 않습니다."
        return true;
    }
    else return false;
}
```

After

```java
public boolean isNotValidCard(int cardNum){
    return this.cardNum != cardNum;
}
```

08 Clean Code – 클래스명 일관화



클래스명의 일관성을 유지하도록 변경

09 OOAD 개발방법론(UP) 느낀 점

Category	Description
장점	- 개발에 이정표가 있어 불필요한 시간 낭비가 없었다. - 정해진 틀을 통해 팀원과 공유하고 의논해 의사소통과 상황공유가 원활하다. - 여러 차례 반복하여 개발의 모호성이 줄어들고 목표가 구체화 되었다.
단점	- 사소한 기능 변화에도 모든 문서를 변경해야 하는 번거로움이 있다. - 디자인 및 문서화 하는 시간이 오래 걸린다. - 이전 단계에 대한 이해가 없으면 다음 단계의 작성에 어려움이 있어 분업이 불가능하다.
개선 할 점	- 문서 작업을 간편화 할 필요가 있다. - 문서 작성을 전담할 인원과 개발 및 구현할 인원을 분리하여 인력과 시간을 효율화 한다.

10 Traceability Tables

Ref. #	Function	usecase	Operation	Method	Class
R 1.1	Select mode 0	Select mode	selection	setVmIp(int vmid, String ip): void	VM
R 1.2	Set up 0	Set up	setVmData	setVmid(int vmid): void	
R 1.3	Select item 0	Select item	setItem	setMid(int mid): void	
R 2.1	msg Request 0	msg Request	selectItem	setCard(int cardNum, int cardBalance)	
R 2.2	msg Return 0	msg Return	sendMsg	setItem(int code, int count, int price): void	
R 3.1	Normal Payment 0	Normal Payment	returnMSg	selectItem(int code, int count): String	
R 3.2	Prepayment 0	Prepayment	checkCard	returnMsg(StringmsgType): ArrayList<Message>	
R 3.3	Check authentication code 0	Check authentication code	checkItemStock	checkItemStock(int code, int count): String	
R 4.1	Check item stock 0	Check item stock	checkAuthCode	normalPayment(int cardNum, int code, int count): boolean	
R 5.1	Check Manager Authentication code 0	Read Manager Authentication code	checkManagerAcess	prePayment(int cardNum, int code, int count, String dstId): String	
R 5.2	Manage stock o	Manage stock	update	findNearVm(ArrayList, int count): String	
				getPosition(): int	
				insertAuthCode(int code, int count, String authCode): void	
				createAuthCode(): String	
				checkManagerAccess(int code): boolean	
				setVmIp(int index, String Ip): void	VMData
				setXYPos(int x, int y): void	
				getVmIp(int i): String	
				setVmid(int i): String	
				getVmIpLength(): int	
				setVmid(int vmid): void	
				getXpos(): int	
				getyPos(): int	
				withdrawBalance(int payAmount): void	CardData
				insNotValidCard(int cardNum): boolean	
				getCardBalance(): int	
				insertAuthCode(int code, int count, String authCode): void	AuthcodeData
				checkAuthCode(String authCode): boolean	
				deleteAuthCode(String authCode): void	
				sendMsg(): void	Message
				setMsgDescription(): String	
				send(String msg): void	
				send(String msg): void	BroadcastMsg
				send(String msg): void	OnedDstMsg
				notifyItem(int code, int count): void	Item
				setItem(int code, int count, int price): void	
				getItemCount(int code): int	
				item_checkItemStock(int code, int count): boolean	
				deleteObserver(Observer): void	
				addObserver(Observer): void	
				updateItemStock(int code, count): void	
				update(int code, int count): void	
				getMid(): int	
				getItemPrice(int code): int	
				update(int code, int count): void	ItemObserver

Q&A

201912698 이현규

202011313 손윤환

202014186 오상희

201711310 변건우