OOPT Second Cycle

4팀: 곽민정, 김재윤,
이승진, 최준

Index

Clean Code

Controller

(before)

```
26 usages
private String dCode;
17 usages
private int count;
21 usages
private DVM myDVM;
5 usages
private MessageManager myMessageManager;
3 usages
private ArrayList<Message> myMessage= new ArrayList<Message>();
```

(after)

```
26 usages
private String drinkCode;
17 usages
private int drinkCount;
21 usages
private DVM myDVM;
5 usages
private MessageManager myMessageManager;
3 usages
private ArrayList<Message> receivedMsgList = new ArrayList<Message>();
3 usages
private String msgTeamID;
```

Clean Code

getClosestDVM()

(before)

```
int x = myLoc.getX();
int y = myLoc.getY();
int min = 9999;
String tempID = " ";
```

(after)

```
public Location getClosestDVM() {
    Location returnLoc = new Location();
    Location myLoc = myDVM.getLocation();
    int myX = myLoc.getX();
    int myY = myLoc.getY();
    int minDistance = 9999;
    String minDvmID = " ";
```

Clean Code

setDrinkKinds()

(before)

```
int count = 0;
int drinkCode = 0;
int[] dCodeArr = new int[7];
```

(after)

```
public void setDrinkKinds() {
    int stockCount = 0;
    int drinkCode = 0;
    int[] salesDrinkCodeArr = new int[MAX_SALESNUM];
```

Clean Code

getOutDrink()

(before)

```
public void getOutDrink(int Calc) {
    int dCode_ = Calc % 100;
    int count_ = Calc / 100;
```

(after)

```
public void getOutDrink(int selectedDrinkInfo) {
    int drinkCode_ = selectedDrinkInfo % 100;
    int drinkCount_ = selectedDrinkInfo / 100;

    Item item = myDVM.getItemList()[drinkCode_ -1];
```

Design Pattern

- Singleton pattern
- Template method pattern

Singleton pattern
(before)

```
public class Controller {

    1 usage   minnnmin +1
    public Controller() {
        myDVM = new DVM();
        myMessageManager = new MessageManager();
    }
```

Singleton pattern
(after)

```
public class MessageManager {
    1 usage  minnmin
    private MessageManager() {}

    3 usages
    private static MessageManager uniqueMessageManager;
    4 usages
    private Serializer mySerializer = new Serializer();

    3 usages  cj +1
    public static MessageManager getInstance() {
        if (uniqueMessageManager == null) {
            uniqueMessageManager = new MessageManager();
        }
        return uniqueMessageManager;
    }
```

```
public class DVM {
    3 usages
    private String id;
    4 usages
    private Location location;
    12 usages
    private Item[] itemList;
    8 usages
    private LinkedList<String> vCodeList = new LinkedList<String>();
    6 usages
    private LinkedList<String> prepayItemList = new LinkedList<String>();
    2 usages
    private String adminPassword;
    3 usages
    private static DVM uniqueDVM;

    2 usages  new *
    public static DVM getInstance() {
        if(uniqueDVM == null)
            uniqueDVM = new DVM();
        return uniqueDVM;
    }
```

Singleton pattern
(after)

```
public class Controller {
    2 usages
    final int MAX_SALESNUM = 7;
    1 usage  Yun531 +1
    public Controller() {
        myDVM = DVM.getInstance();
        myMessageManager = MessageManager.getInstance();
        paymentPage = new PaymentPage();
        prePaymentPage = new PrePaymentPage();
        verificationCodeMenu = new VerificationCodeMenu();

    }
```

```
public abstract class Pay {
    3 usages
    Scanner scan = new Scanner(System.in);
    4 usages
    DVM myDVM = DVM.getInstance();
    5 usages
    int totalPrice;
    13 usages
    String Info;
    4 usages
    String dCode;
    4 usages
    int count;
    2 usages
    String dstID;
```

Template method pattern
(abstract class Pay)

```
void pay(String dCode, int count, int totalPrice, String dstID){
    this.totalPrice = totalPrice;
    this.dCode = dCode;
    this.count = count;
    this.dstID = dstID;

    showPage();
    int errno = inputInfo();  //-1: 비정상 입력, 0: 입력 종료, 1: 정상 입력
    proceedPay(errno);
}
```

```
void proceedPay(int errno){
    switch(errno){
        case -1:
            printErr();
            break;
        case 0:
            printExit();
            break;
        case 1:
            payment();
            break;
    }
}
```

Template method pattern

(PaymentPage extends Pay)

```
void showPage(){
    System.out.println("<결제>");
    System.out.println("(메뉴 선택으로 돌아가려면 \"0\"을 입력해주세요)\n");
    System.out.println("카드 번호를 입력하세요.");
    System.out.print(">");
}
```

```
int inputInfo(){

    this.Info =scan.nextLine();

    if(this.Info.equals("0")) {
        return 0; //showMenu로 돌아감
    }
    else if(CardCompany.isValidCard(this.Info,totalPrice)){
        return 1;
    }
    else {
        return -1; //showMenu로 돌아감
    }
}
```

Template method pattern

(PrePaymentPage extends Pay)

```
void showPage(){
    System.out.println("<결제>");
    System.out.println("(메뉴 선택으로 돌아가려면 \"0\"을 입력해주세요)\n");
    System.out.println("카드 번호를 입력하세요.");
    System.out.print(">");
}
```

```
int inputInfo(){

    this.Info =scan.nextLine();

    if(this.Info.equals("0")) {
        return 0; //showMenu로 돌아감
    }
    else if(CardCompany.isValidCard(this.Info,totalPrice)){
        return 1;
    }
    else {
        return -1; //showMenu로 돌아감
    }
}
```

Template method pattern

(VerificationCodeMenu extends Pay)

```
void showPage(){
    System.out.println("선결제 후 받은 인증코드를 입력해주세요\n" +
            "(메뉴 선택으로 돌아가려면 “0”를 입력해주세요)\n");
    System.out.print(">");
}
```

```
int inputInfo(){
    this.Info=scan.nextLine();

    if(Info.equals("0")) {
        return 0; //showMenu로 돌아감
    }
    else if(isRightVerificationCode(Info) && myDVM.isValidVerificationCode(Info)){
        return 1;
    }
    else {
        return -1; //showMenu로 돌아감
    }
}
```

Template method pattern

(before)

```
scan.nextLine();
if(myDVM.checkStock(Integer.parseInt(dCode), count)) {
    showPaymentPage(calculateTotalPrice());
} else {
    if(showOtherDVM(getClosestDVM())) {
        showPaymentPage(calculateTotalPrice());

        String UserVCode;
        UserVCode = createVerificationCode();
        myMessageManager.sendReqMsg( type: "PrepaymentCheck", dCode, count, UserVCode);
        System.out.println("선결제가 완료되었습니다.\n" +
                "인증코드: " + UserVCode);
    }
    else{
        System.out.println("해당 음료에 대한 재고를 보유한 DVM이 존재하지 않습니다.");
    }
}
```

(after)

```
scan.nextLine();
if(myDVM.checkStock(Integer.parseInt(dCode), count)) {
    paymentPage.pay(dCode, count, calculateTotalPrice(),  dstID: "0");
} else {

    if(showOtherDVM(getClosestDVM())) {
        System.out.print("안내된 DVM에서 선결제를 진행하시겠습니까?\n" +
                "(1: 선결제 진행, 나머지 정수: 진행 취소\n" +
                ">");
        int mode;
        while (true) {
            try {
                mode = scan.nextInt();
                break;
            } catch (InputMismatchException ime) {
                scan.next();
                System.out.println("잘못된 입력입니다. 정수만 입력해주세요.");
                System.out.print("안내된 DVM에서 선결제를 진행하시겠습니까?\n" +
                        "(1: 선결제 진행, 나머지 정수: 진행 취소\n" +
                        ">");
            }
        }
        scan.nextLine();
        if(mode == 1) {
            prePaymentPage.pay(dCode, count, calculateTotalPrice(), dstID);
        }
        else{
            System.out.println("선결제를 진행하지 않고, 메뉴 선택으로 돌아갑니다.");
        }
    }
    else{
        System.out.println("해당 음료에 대한 재고를 보유한 DVM이 존재하지 않습니다.");
    }
}
```

Template method pattern

(before)

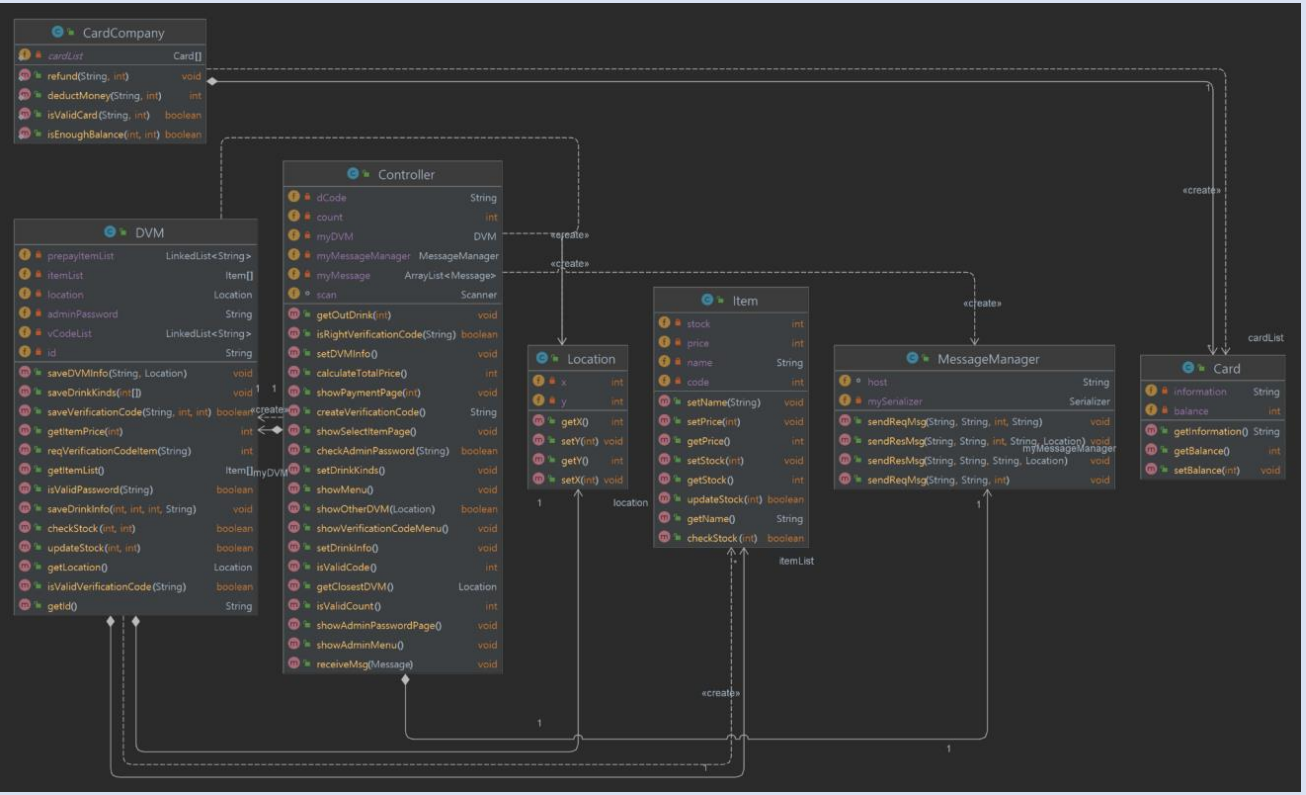
```
public void showVerificationCodeMenu() {
    String vCode = "입력오류";
    boolean vCodeTR = false;

    while(true) {
        System.out.println("선결제 후 받은 인증코드를 입력해주세요\n" +
                "(메뉴 선택으로 돌아가려면 "0"를 입력해주세요)\n");
        System.out.print(">");

        try {
            vCode = scan.next();
        } catch(InputMismatchException ime) {
            System.out.println("잘못된 입력입니다.");
            continue;
        }

        if(isRightVerificationCode(vCode)) {          //인증코드 입력형식 test
            break;
        }
        System.out.println("입력하신 인증코드가 입력형식에 맞지 않습니다.\n");

    }

    vCodeTR = myDVM.isValidVerificationCode(vCode);             //유효한 인증코드 test
    if(vCode.equals("0")) {

    } else if (!vCodeTR) {
        System.out.println("유효하지 않은 인증코드입니다.");
    } else {
        getOutDrink(myDVM.reqVerificationCodeItem(vCode));
    }
}
```

(after)

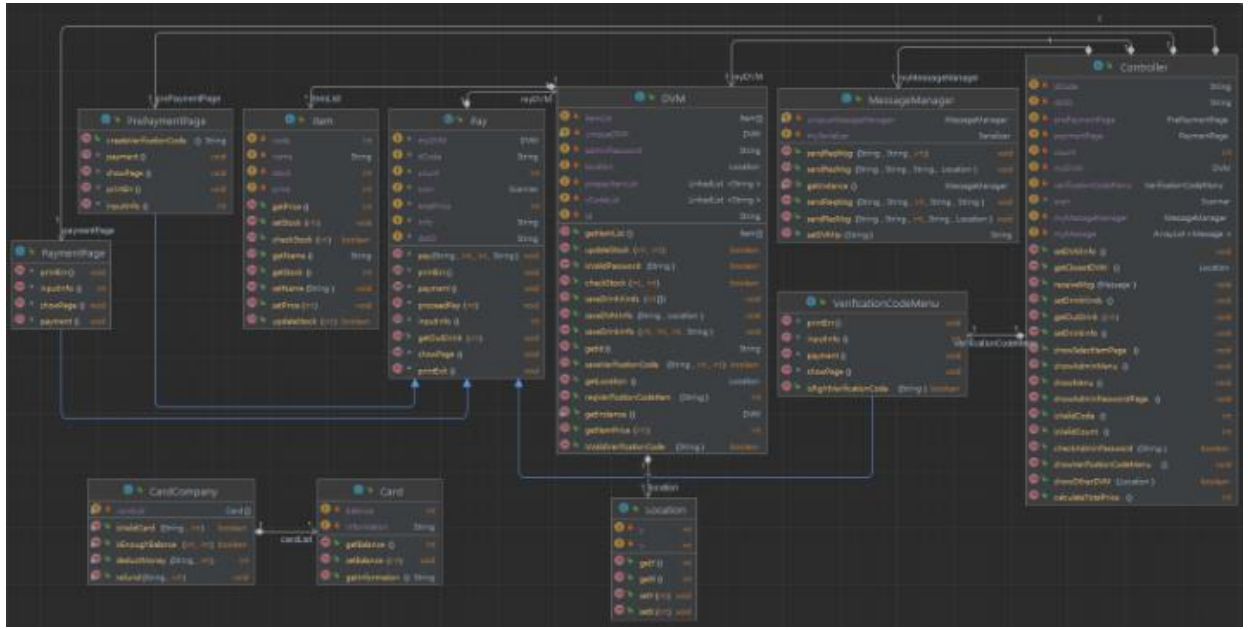
```
public void showVerificationCodeMenu() {
    verificationCodeMenu.pay( dCode: "1",  count: 0,  totalPrice: 0,  dstID: "0");
}
```

Template method pattern

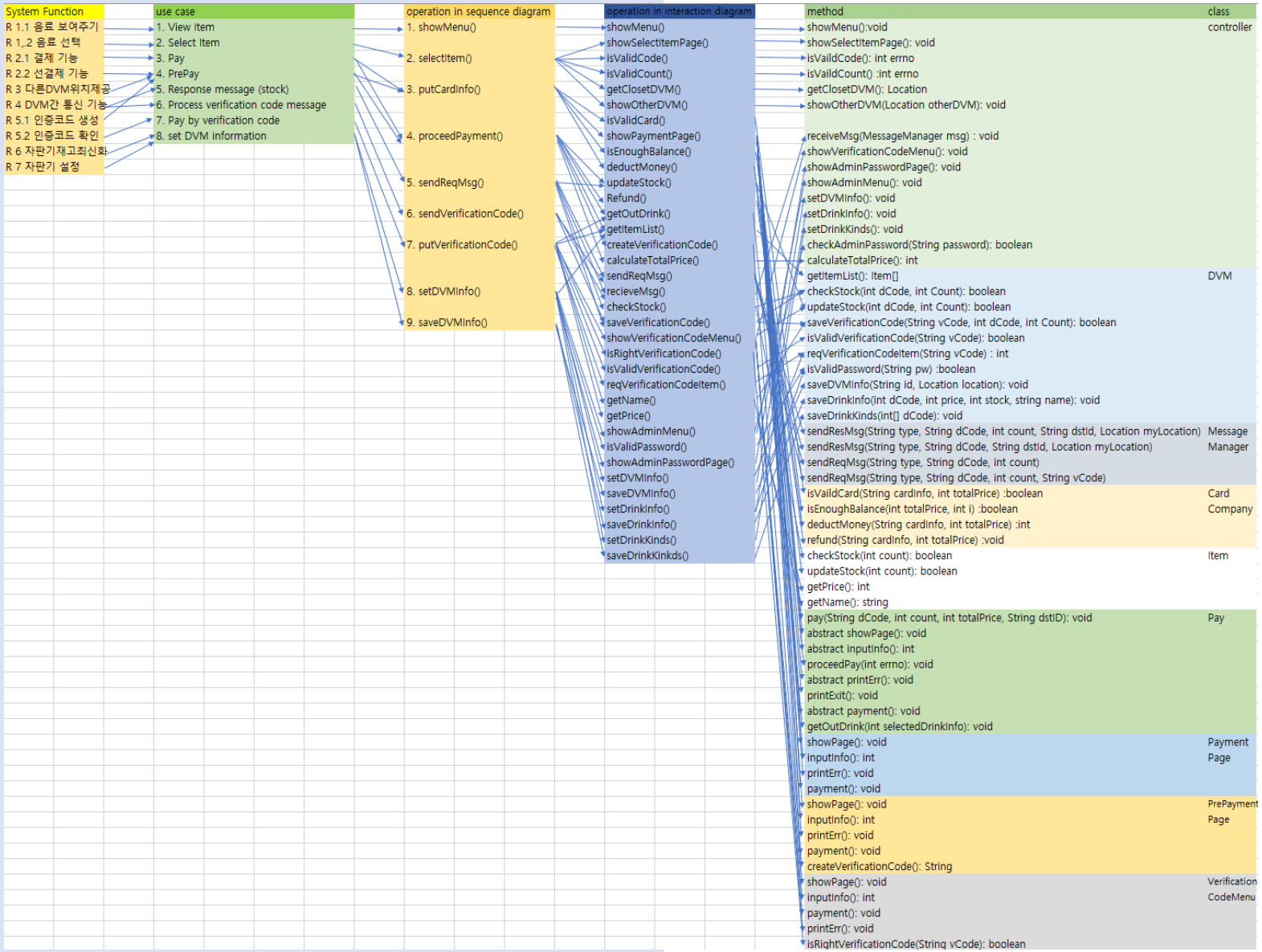
(before)



(after)

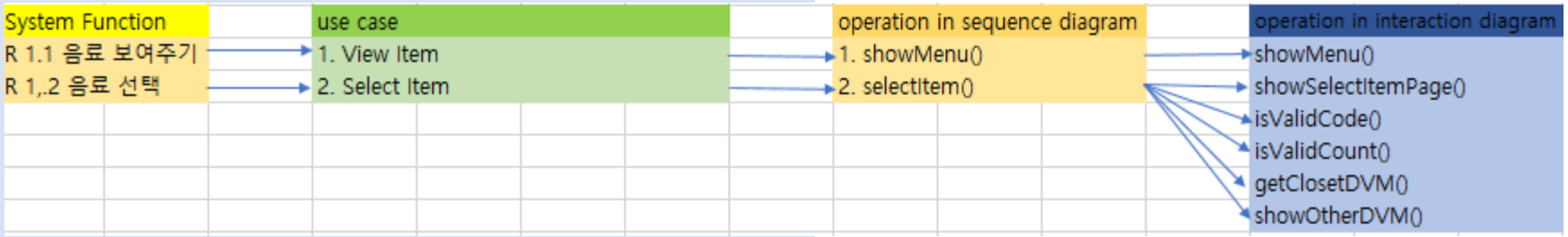


Traceability

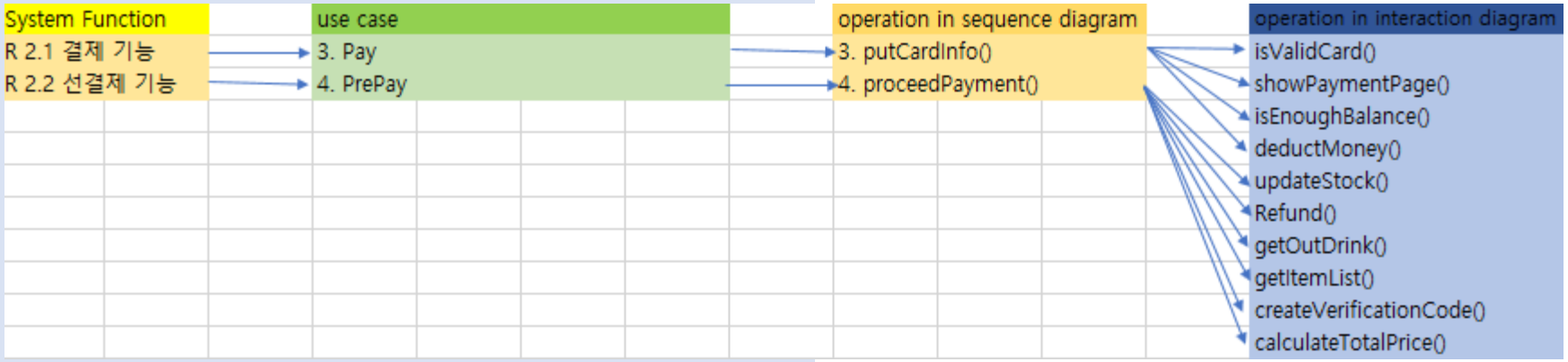


System Function	use case	operation in sequence diagram	operation in interaction diagram	method	class
R 1.1 음료 보여주기	1. View Item	1. showMenu()	showMenu()	showMenu():void	controller
R 1,.2 음료 선택	2. Select Item	2. selectItem()	showSelectItemPage()	showSelectItemPage(): void	
R 2.1 결제 기능	3. Pay	3. putCardInfo()	isValidCode()	isVaildCode(): int errno	
R 2.2 선결제 기능	4. PrePay	4. proceedPayment()	isValidCount()	isVaildCount() :int errno	
R 3 다른DVM위치제공	5. Response message (stock)	5. sendReqMsg()	getClosetDVM()	getClosetDVM(): Location	
R 4 DVM간 통신 기능	6. Process verification code message	6. sendVerificationCode()	showOtherDVM()	showOtherDVM(Location otherDVM): void	
R 5.1 인증코드 생성	7. Pay by verification code	7. putVerificationCode()	isValidCard()		
R 5.2 인증코드 확인	8. set DVM information	8. setDVMInfo()	showPaymentPage()	receiveMsg(MessageManager msg) : void	
R 6 자판기재고최신화		9. saveDVMInfo()	isEnoughBalance()	showVerificationCodeMenu(): void	
R 7 자판기 설정			deductMoney()	showAdminPasswordPage(): void	
			updateStock()	showAdminMenu(): void	
			Refund()	setDVMInfo(): void	
			getOutDrink()	setDrinkInfo(): void	
			getItemList()	setDrinkKinds(): void	
			createVerificationCode()	checkAdminPassword(String password): boolean	
			calculateTotalPrice()	calculateTotalPrice(): int	
			sendReqMsg()	getItemList(): Item[]	DVM
			recieveMsg()	checkStock(int dCode, int Count): boolean	
			checkStock()	updateStock(int dCode, int Count): boolean	
			saveVerificationCode()	saveVerificationCode(String vCode, int dCode, int Count): boolean	
			showVerificationCodeMenu()	isValidVerificationCode(String vCode): boolean	
			isRightVerificationCode()	reqVerificationCodeItem(String vCode) : int	
			isValidVerificationCode()	isValidPassword(String pw) :boolean	
			reqVerificationCodeItem()	saveDVMInfo(String id, Location location): void	
			getName()	saveDrinkInfo(int dCode, int price, int stock, string name): void	
			getPrice()	saveDrinkKinds(int[] dCode): void	
			showAdminMenu()	sendResMsg(String type, String dCode, int count, String dstId, Location myLocation)	Message
			isValidPassword()	sendResMsg(String type, String dCode, String dstId, Location myLocation)	Manager
			showAdminPasswordPage()	sendReqMsg(String type, String dCode, int count)	
			setDVMInfo()	sendReqMsg(String type, String dCode, int count, String vCode)	
			saveDVMInfo()	isVaildCard(String cardInfo, int totalPrice) :boolean	Card
			setDrinkInfo()	isEnoughBalance(int totalPrice, int i) :boolean	Company
			saveDrinkInfo()	deductMoney(String cardInfo, int totalPrice) :int	
			setDrinkKinds()	refund(String cardInfo, int totalPrice) :void	
			saveDrinkKinkds()	checkStock(int count): boolean	Item
				updateStock(int count): boolean	
				getPrice(): int	
				getName(): string	
				pay(String dCode, int count, int totalPrice, String dstID): void	Pay
				abstract showPage(): void	
				abstract inputInfo(): int	
				proceedPay(int errno): void	
				abstract printErr(): void	
				printExit(): void	
				abstract payment(): void	
				getOutDrink(int selectedDrinkInfo): void	
				showPage(): void	Payment
				inputInfo(): int	Page
				printErr(): void	
				payment(): void	
				showPage(): void	PrePayment
				inputInfo(): int	Page
				printErr(): void	
				payment(): void	
				createVerificationCode(): String	
				showPage(): void	Verification
				inputInfo(): int	CodeMenu
				payment(): void	
				printErr(): void	
				isRightVerificationCode(String vCode): boolean	

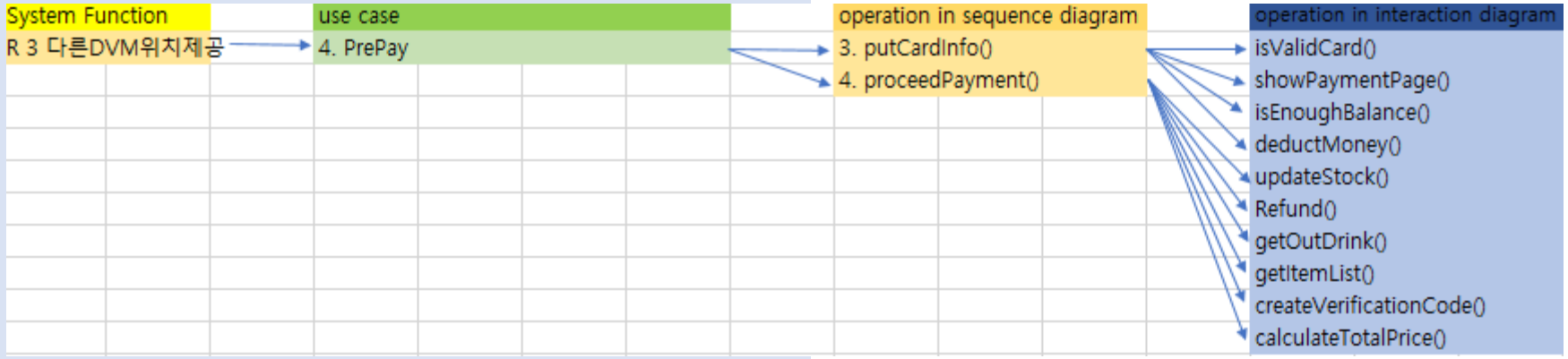
Traceability



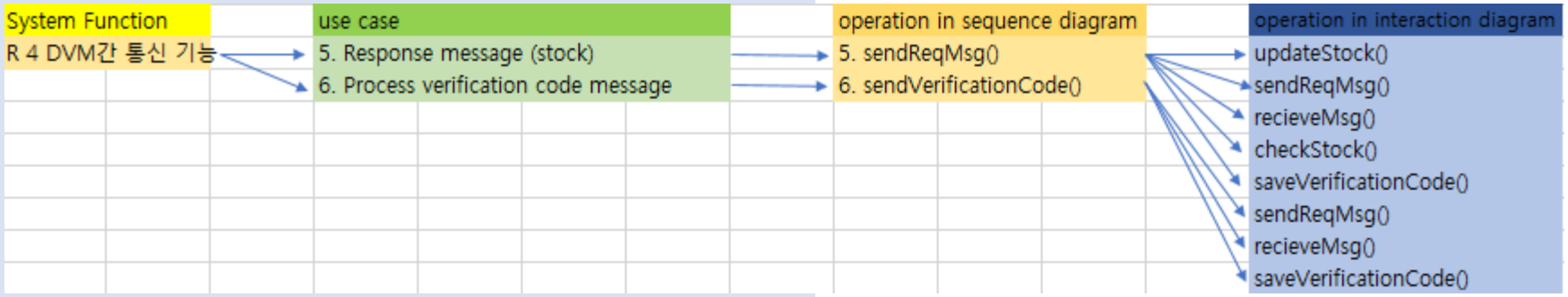
Traceability



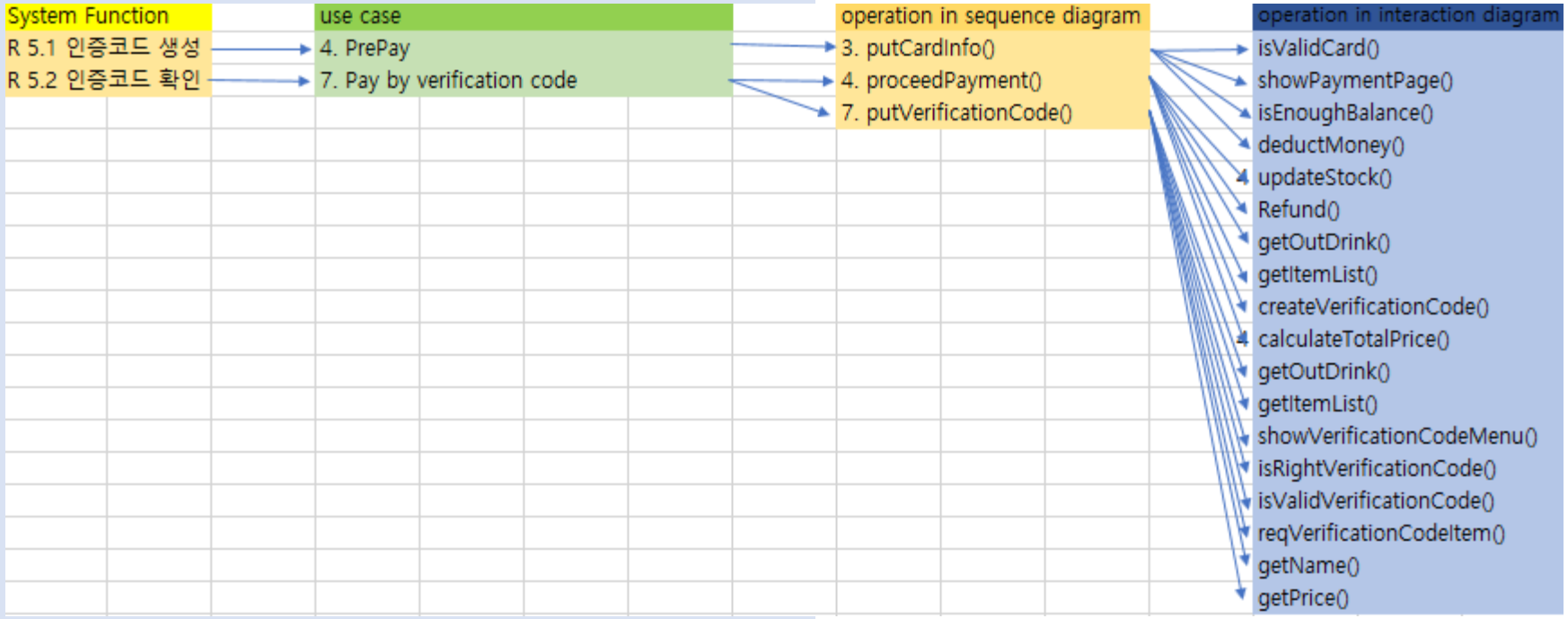
Traceability



Traceability



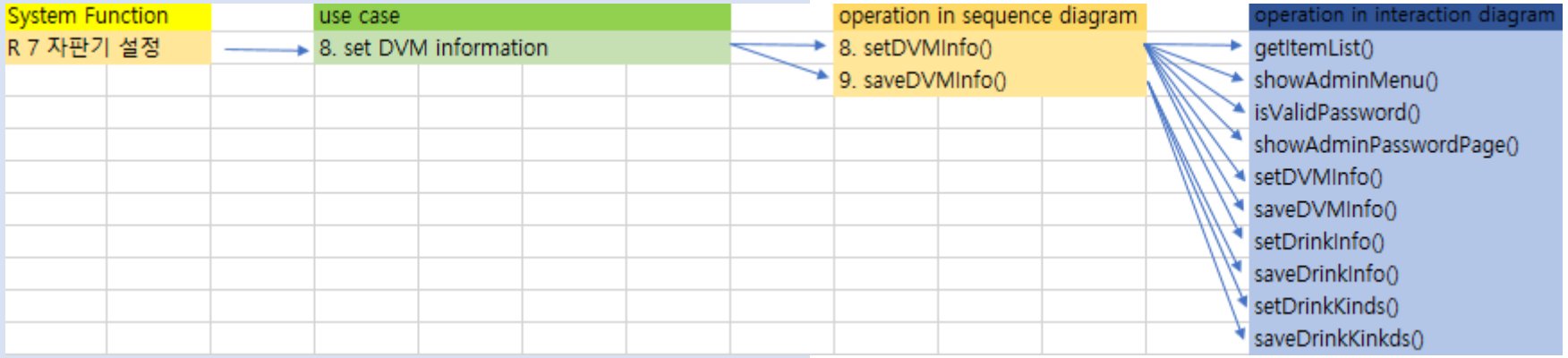
Traceability



Traceability

System Function	use case	operation in sequence diagram	operation in interaction diagram
R 6 자판기재고최신화	8. set DVM information	8. setDVMInfo()	getItemList()
		9. saveDVMInfo()	showAdminMenu()
			isValidPassword()
			showAdminPasswordPage()
			setDVMInfo()
			saveDVMInfo()
			setDrinkInfo()
			saveDrinkInfo()
			setDrinkKinds()
			saveDrinkKinkds()

Traceability



UP 개발 방식에 대한 소감(장점)

- 곽민정: 예쁜 코드, 수정 용이성

  디버깅 시 어느 부분을 고쳐야 하는지, 필요한 것을 어떤 클래스에 추가해야 하는지 등을 쉽게 알 수 있음.

  구현 전까지 꼼꼼한 설계를 하고, 또 tool을 사용해 스켈레톤 코드를 자동으로 생성할 수 있기 때문에 직접 코딩을 할 내용이 적음.

  iteration을 여러 번 돌기 때문에, 잘 해야 한다는 부담이 적음

- 김재윤: 개발방법론을 사용하지 않고 주먹구구식으로 개발을 진행할 때에는 개발과정이 매끄럽지 않았지만, 해당 개발방법론을 사용할 경우 개발과정이 굉장히 매끄러워 졌다

- 이승진: 기획, 설계하는 사람과 구현하는 사람이 달라도 문제가 없음, 설계만 잘 되어 있다면 누가 구현해도 동일한 결과를 이끌어 낼 수 있다.

- 최준:

  Class diagram을 통해 skeleton code를 뽑아내고, 그 코드에서 sequence diagram을 보고 method들을 채워 나가기 때문에 코드 구현이 매우 간단하고 어렵지 않았다.

  FR → use case → sequence diagram으로 오면서 반드시 수행해야 하는 functional requirement들을 놓치지 않고 구현할 수 있었다.

UP 개발 방식에 대한 소감(단점)

- 곽민정:
  분석 및 설계 단계에서 작성할 문서가 많음.
  코드가 길어짐. (간접 호출이 잦고 (직접 해도 될 것을 여러 클래스를 통해 호출한다든지), 클래스가 많기 때문)
- 김재윤:
  최신 작업에서 변경점이 생길 경우 이전 작업의 문서도 같이 변경하여 특정 단계의 모든 문서가 일치하도록 변경해야하는 수고로움이 있음.
  문서화 작업 진행 시, 모든 조원이 참여해서 회의를 하기 때문에 시간적 제약을 많이 받음.
- 이승진: 작성해야할 문서와 diagram이 많으며 이에 필요한 배경지식이 많음.
- 최준:
  코드 구현 시간 대신 문서 작성 시간이 매우 길어진다.
  각 단계별로 문서를 작성하고 다음 단계에서 전 문서를 참고해야 한다. 또 다음 단계 진행 중 전 단계에 수정할 것들을 계속 수정해야 하는 번거로움이 있었다.

적용 가능성, 개선점

- 적용 가능성:

  Class diagram, sequence diagram을 통해 객체 간의 communication들을 명확히 볼 수 있어 객체 지향 언어 외에도 안드로이드 개발에도 적용할 수 있을 것 같다.

  문서가 많이 작성되고 남아있기 때문에 유지보수가 중요한 곳에 적용될 수 있을 것 같다.

- 개선점:

  UML 도구와 문서화 도구가 연결되어 있어, 특정 UML 변경 시 연결된 모든 문서의 해당 UML이 변경되는 새로운 도구가 있으면 문서의 변경사항 추적에 용의할 것 같다.

OOAD를 수행하는 팀 프로젝트가 효과적으로 진행되기 위한 선결조건 및 지원방법

- 분석 단계에서 도메인 전문가 필요
- 팀원 개개인의 OOAD 방법론의 전체적인 프로세스의 이해
- 각 단계에서 필요한 문서, diagram의 notation을 잘 이해하고 있어야 함.
- OOAD에 능숙한 리더의 꾸준한 작업물을 점검 필요
- 문서의 버전 관리 및 변경사항 추적 필요
- smart한 UML 및 code generation tool 제공 - 변수 타입 등을 잘 인식해서, 컴파일 에러로 인한 가벼운 디버깅 하는 시간 낭비를 하지 않도록 해야 함

감사합니다