DISTRIBUTED VENDING MACHINE

객체지향개발방법론 [T2]

- 201711391 류근범
- 202282050 조문충
- 201711436 홍윤표
- 202013557 하지라

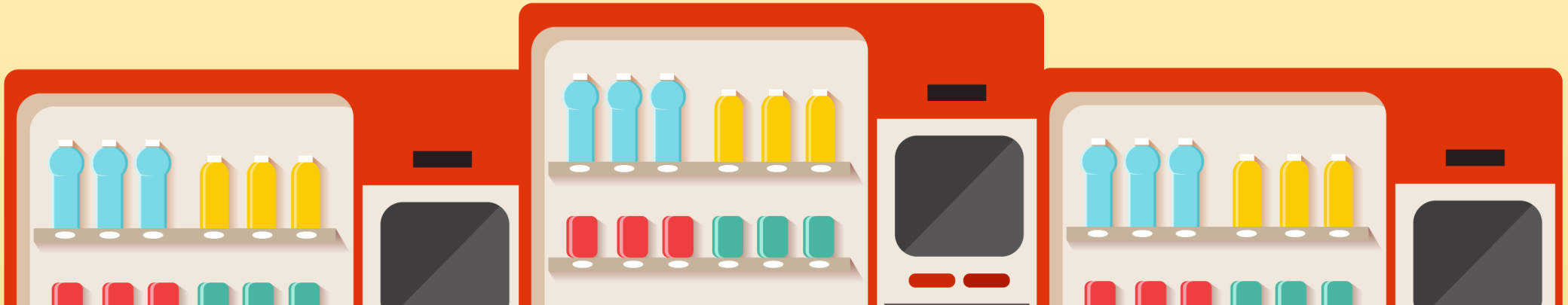
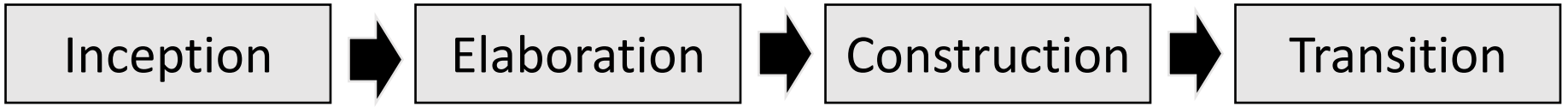


TABLE OF CONTENTS

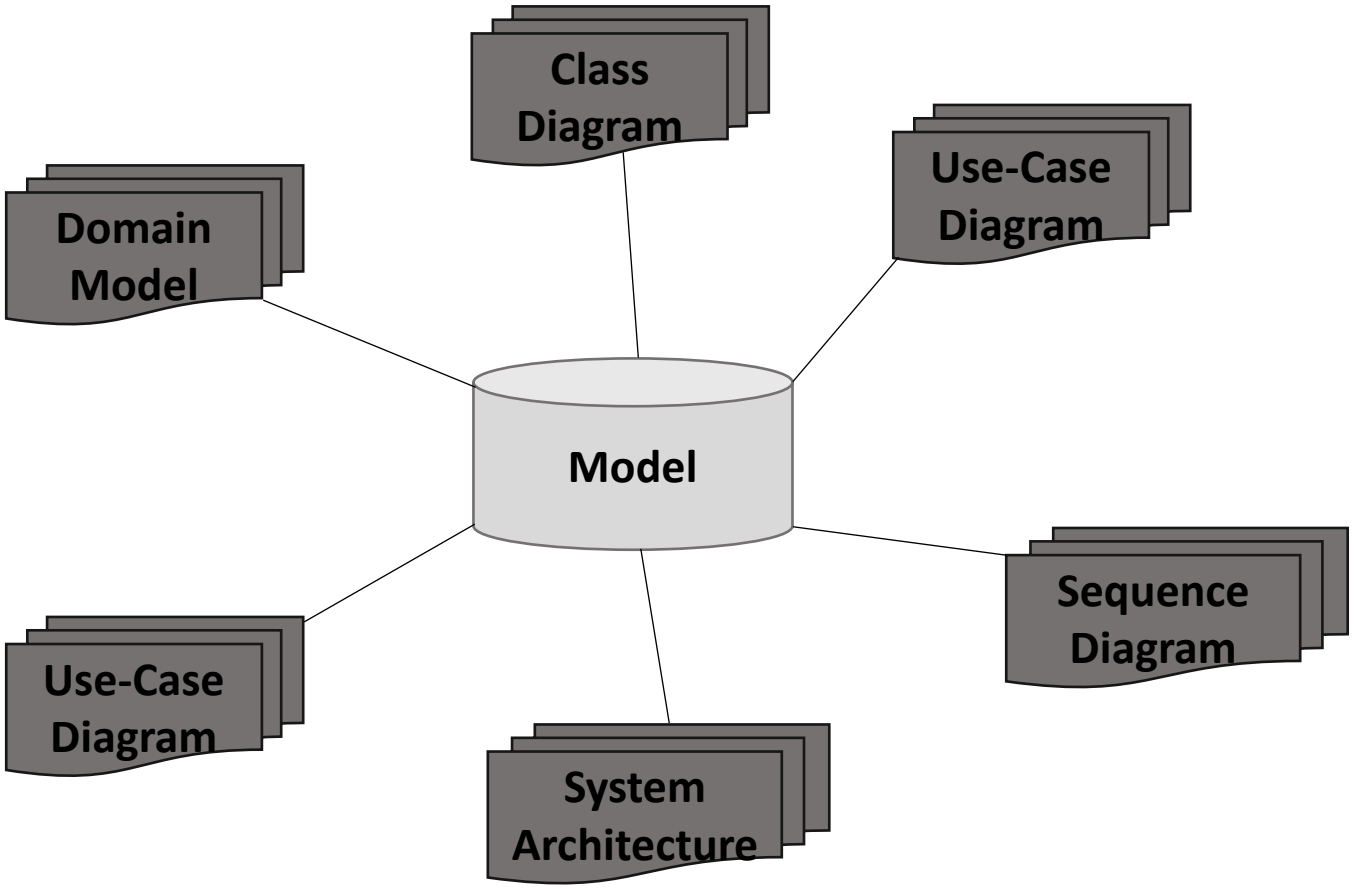
OOPT Final Development Demonstration

- Rational Unified Process Phases
  - Inception
  - Elaboration
  - Construction
  - Transition
- Design Pattern and Clean Code
- Why do we need OOAD(UP)?

Rational Unified Process



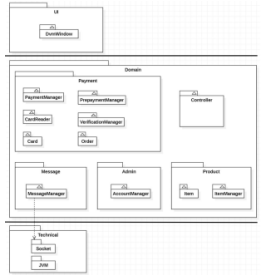
Rational Unified Process

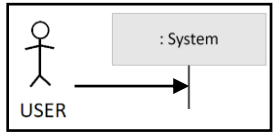


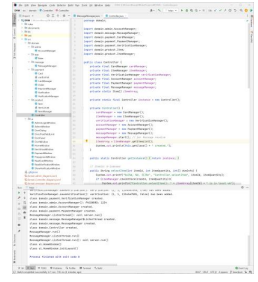
Rational Unified Process

Inception → Elaboration → Construction → Transition

Functional Requirement

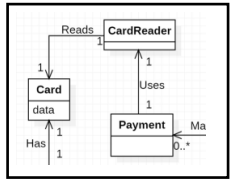


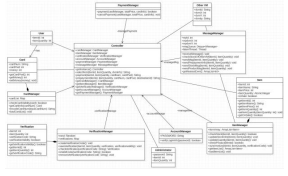


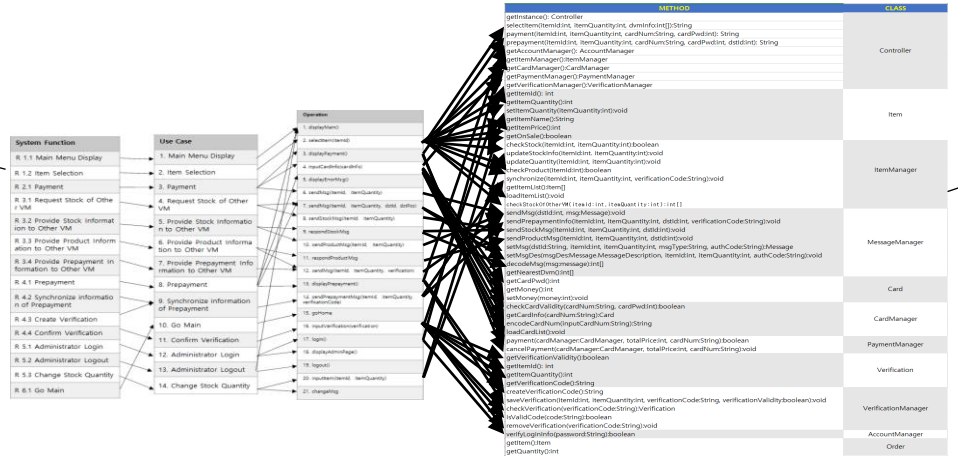


Test-Case

Use-Case







Rational Unified Process

Inception → Elaboration → Construction → Transition

Motivation

NonFunctional

Functional Requirement

Cost

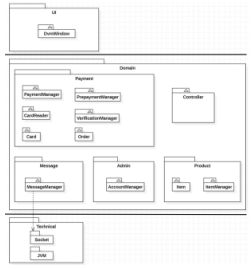
Human Resources

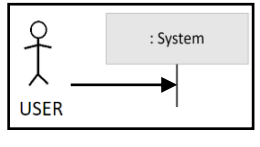
Manage ment

Use-Case

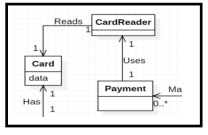
Justification

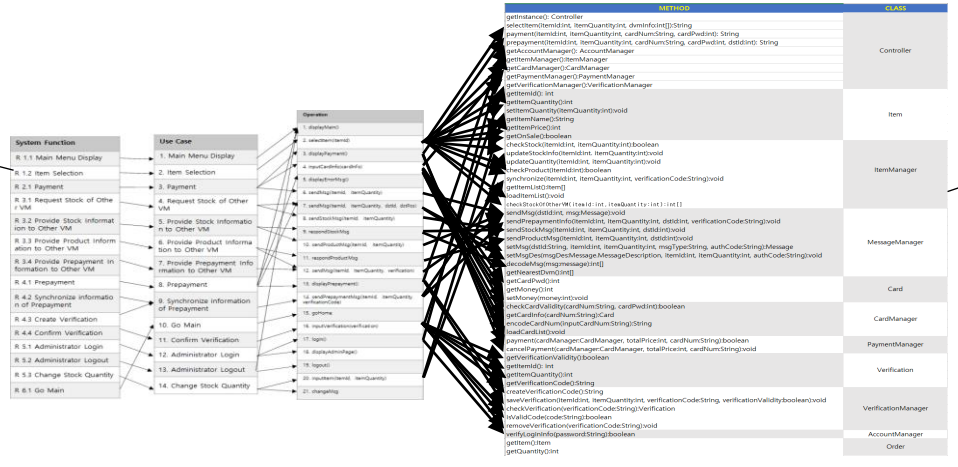
Market Analysis





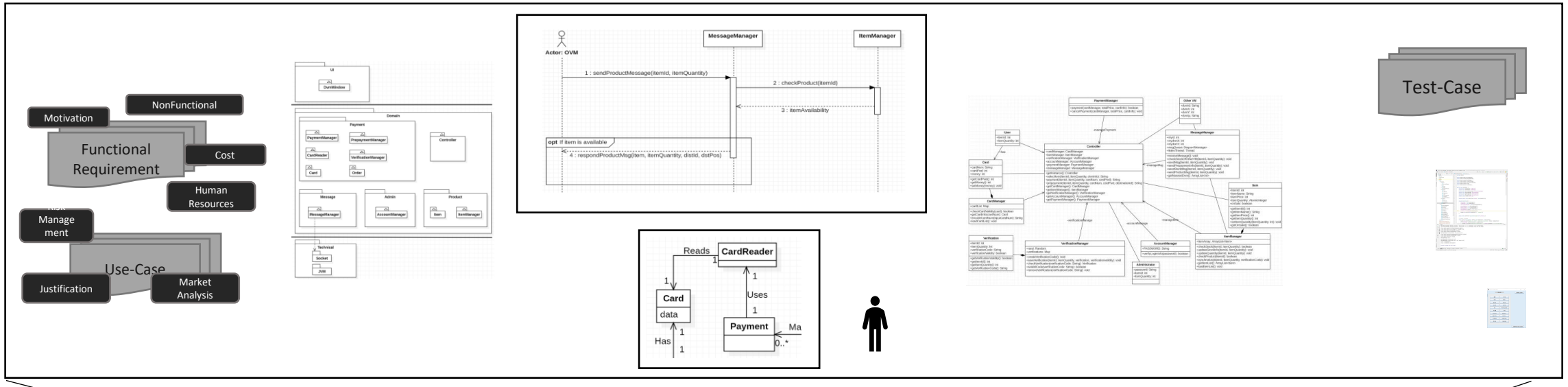
Test-Case

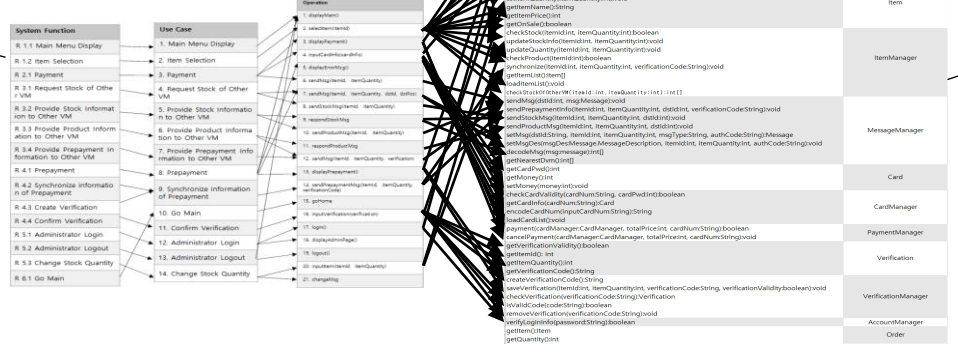




Rational Unified Process

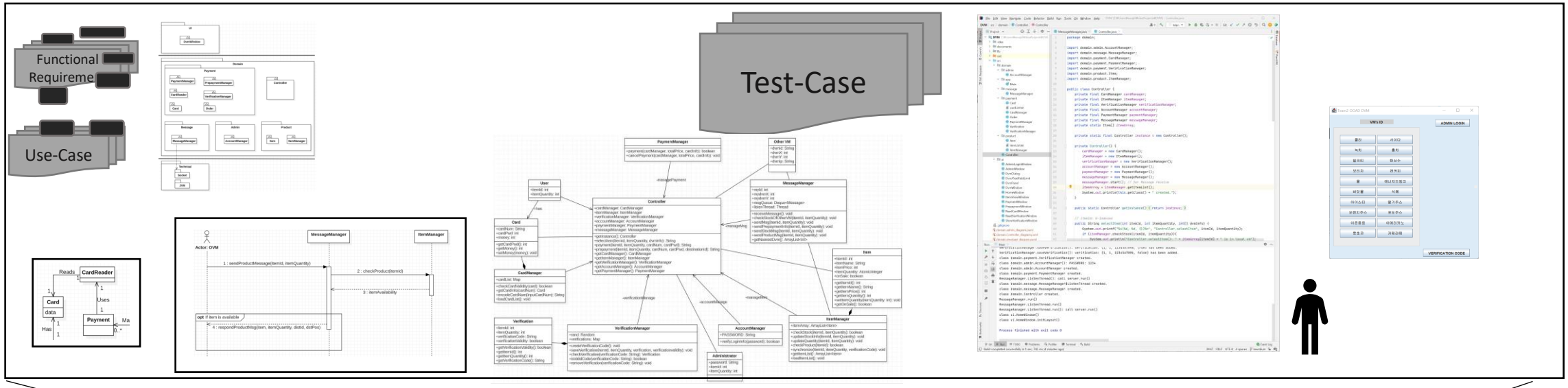
Inception → **Elaboration** → Construction → Transition

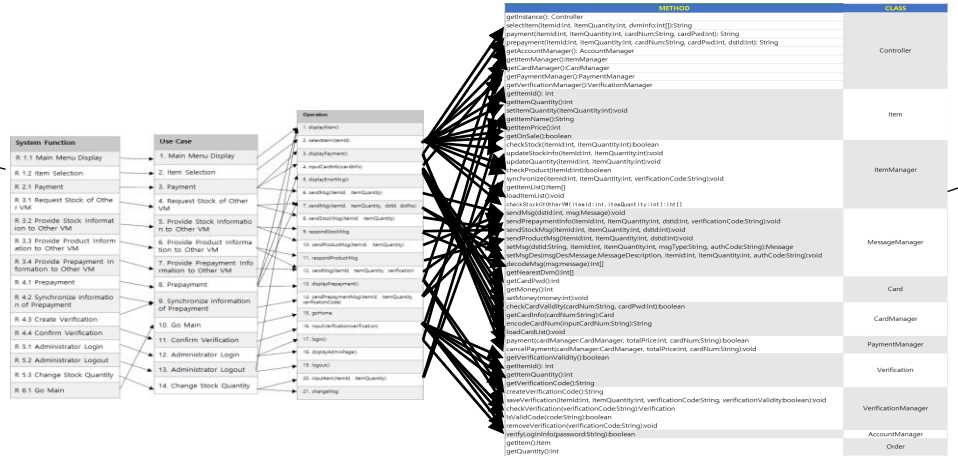




Rational Unified Process

Inception → Elaboration → **Construction** → Transition



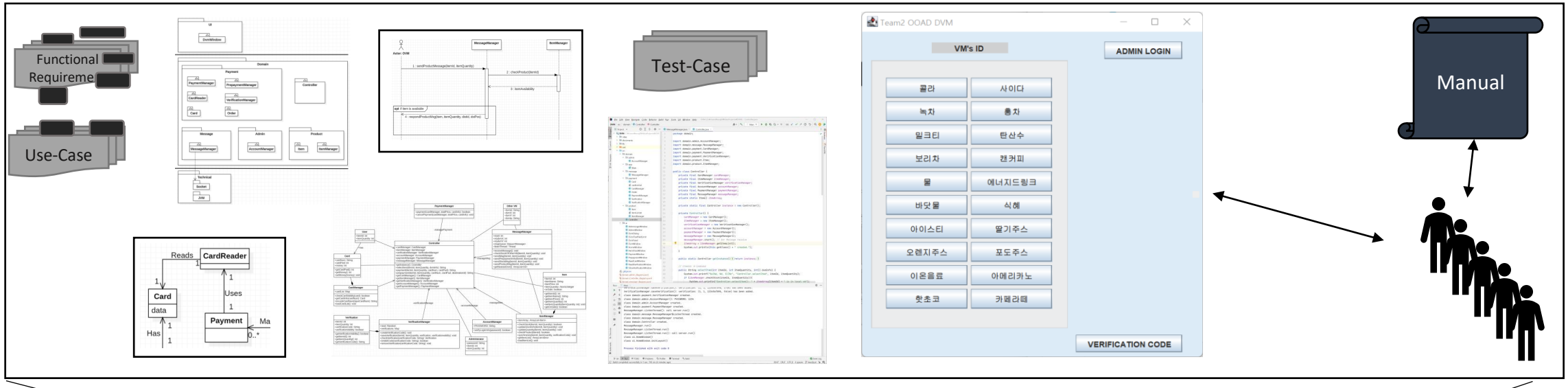


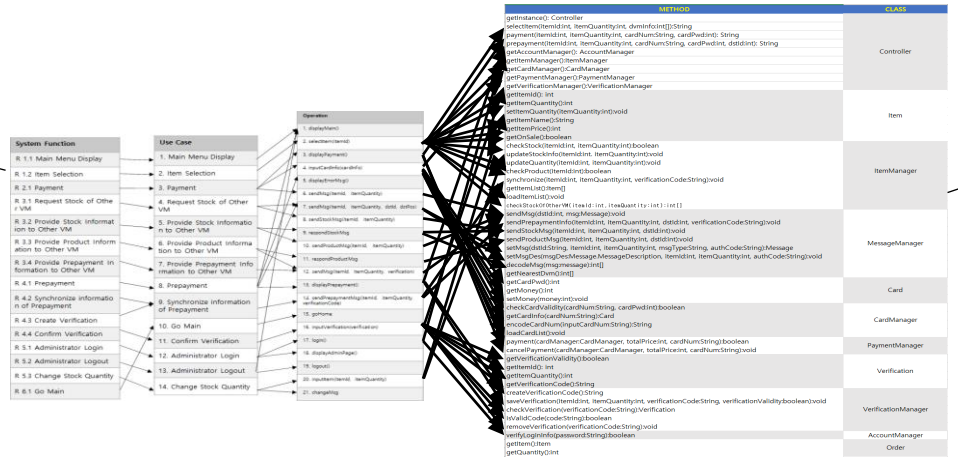
System Function	Use Case
R 1.1 Main Menu Display	1. Main Menu Display
R 1.2 Item Selection	2. Item Selection
R 2.1 Payment	3. Payment
R 3.1 Request Stock of Other VM	4. Request Stock of Other VM
R 3.2 Provide Stock information to Other VM	5. Provide Stock Information to Other VM
R 3.3 Provide Product Information to Other VM	6. Provide Product Information to Other VM
R 3.4 Provide Prepayment information to Other VM	7. Provide Prepayment Information to Other VM
R 4.1 Prepayment	8. Prepayment
R 4.2 Synchronize information of Prepayment	9. Synchronize Information of Prepayment
R 4.3 Create Verification	10. Go Main
R 4.4 Confirm Verification	11. Confirm Verification
R 5.1 Administrator Login	12. Administrator Login
R 5.2 Administrator Logout	13. Administrator Logout
R 5.3 Change Stock Quantity	14. Change Stock Quantity
R 6.1 Go Main	

METHOD	CLASS
getInstance(): Controller	Controller
selectItem(ItemId:int, ItemQuantity:int, dvmInfoInt[]):String	
payment(ItemId:int, ItemQuantity:int, cardNumString, cardPwd:int): String	
prepayment(ItemId:int, ItemQuantity:int, cardNumString, cardPwd:int, distId:int): String	
getAccountManager(): AccountManager	
getItemManager():ItemManager	
getCardManager():CardManager	
getPaymentManager():PaymentManager	
getVerificationManager():VerificationManager	
getItemId(): int	Item
getItemQuantity():int	
setItemQuantity(itemQuantity:int):void	
getItemName():String	
getItemPrice():int	
getOnSale():boolean	
checkStock(itemId:int, itemQuantity:int):boolean	ItemManager
updateStockInfo(itemId:int, itemQuantity:int):void	
updateQuantity(itemId:int, itemQuantity:int):void	
checkProduct(itemId:int):boolean	
synchronize(itemId:int, itemQuantity:int, verificationCode:String):void	
getItemList():Item[]	
loadItemList():void	
checkStockOtherVM(itemId:int, itemQuantity:int):int[]	
sendMsg(distId:int, msgMessage):void	MessageManager
sendPrepaymentInfo(itemId:int, itemQuantity:int, distId:int, verificationCode:String):void	
sendStockMsg(itemId:int, itemQuantity:int, distId:int):void	
sendProductMsg(itemId:int, itemQuantity:int, distId:int):void	
setMsg(distId:String, itemId:int, itemQuantity:int, msgType:String, authCodeString):Message	
setMsgDes(msgDesMessage:MessageDescription, itemId:int, itemQuantity:int, authCode:String):void	
decodeMsg(msgmessage):int[]	
getNearestDvm():int[]	
getCardPwd():int	Card
getMoney():int	
setMoney(money:int):void	
checkCardValidity(cardNum:String, cardPwd:int):boolean	CardManager
getCardInfo(cardNum:String):Card	
encodeCardNum(inputCardNum:String):String	
loadCardList():void	
payment(cardManager:CardManager, totalPrice:int, cardNum:String):boolean	PaymentManager
cancelPayment(cardManager:CardManager, totalPrice:int, cardNumString):void	
getVerificationValidity():boolean	Verification
getItemId(): int	
getItemQuantity():int	
getVerificationCode():String	
createVerificationCode():String	VerificationManager
saveVerification(itemId:int, itemQuantity:int, verificationCodeString, verificationValidity:boolean):void	
checkVerification(verificationCode:String):Verification	
isValidCode(code:String):boolean	
removeVerification(verificationCode:String):void	
verifyLoginInfo(password:String):boolean	AccountManager
getItem():Item	Order
getQuantity():int	

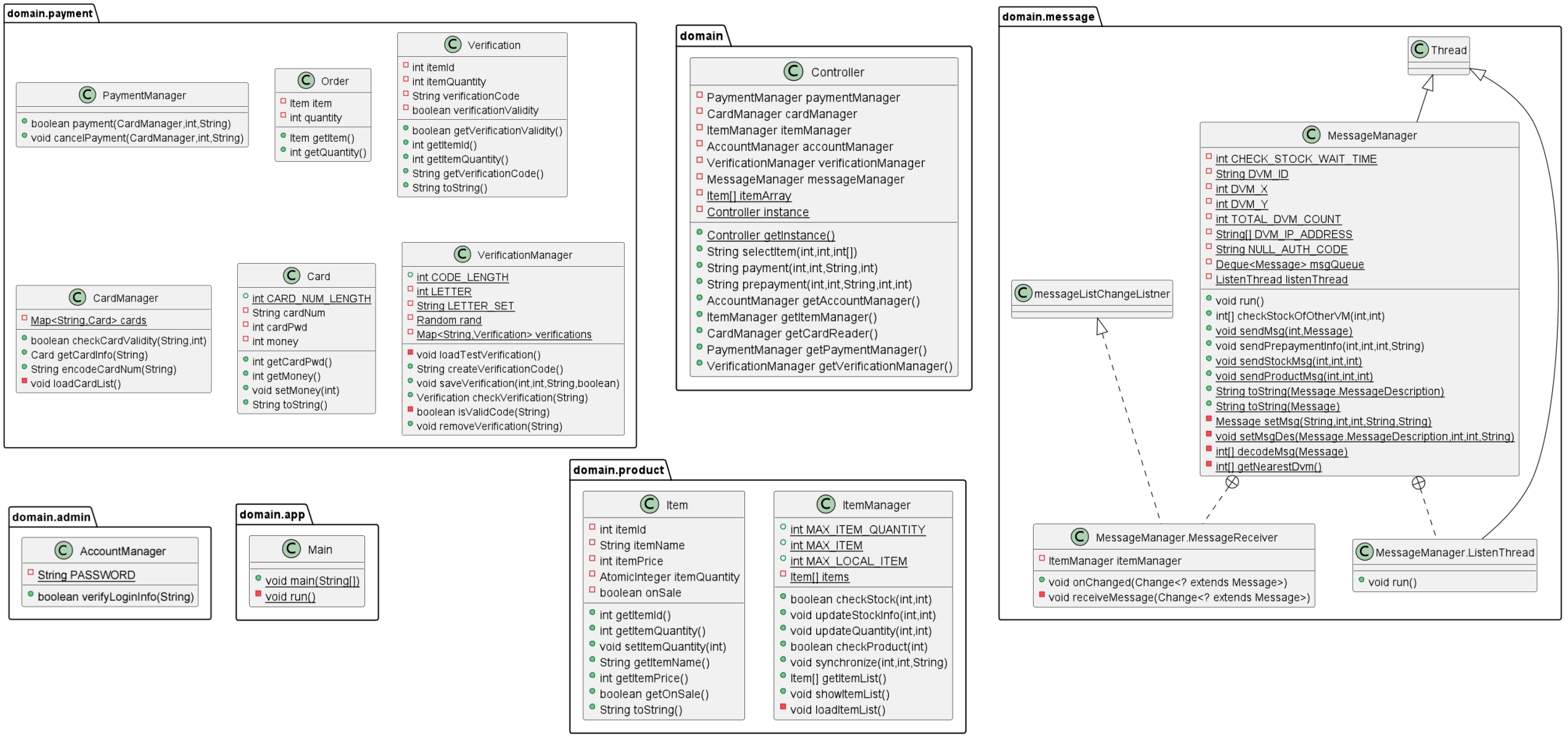
Rational Unified Process

Inception → Elaboration → Construction → Transition

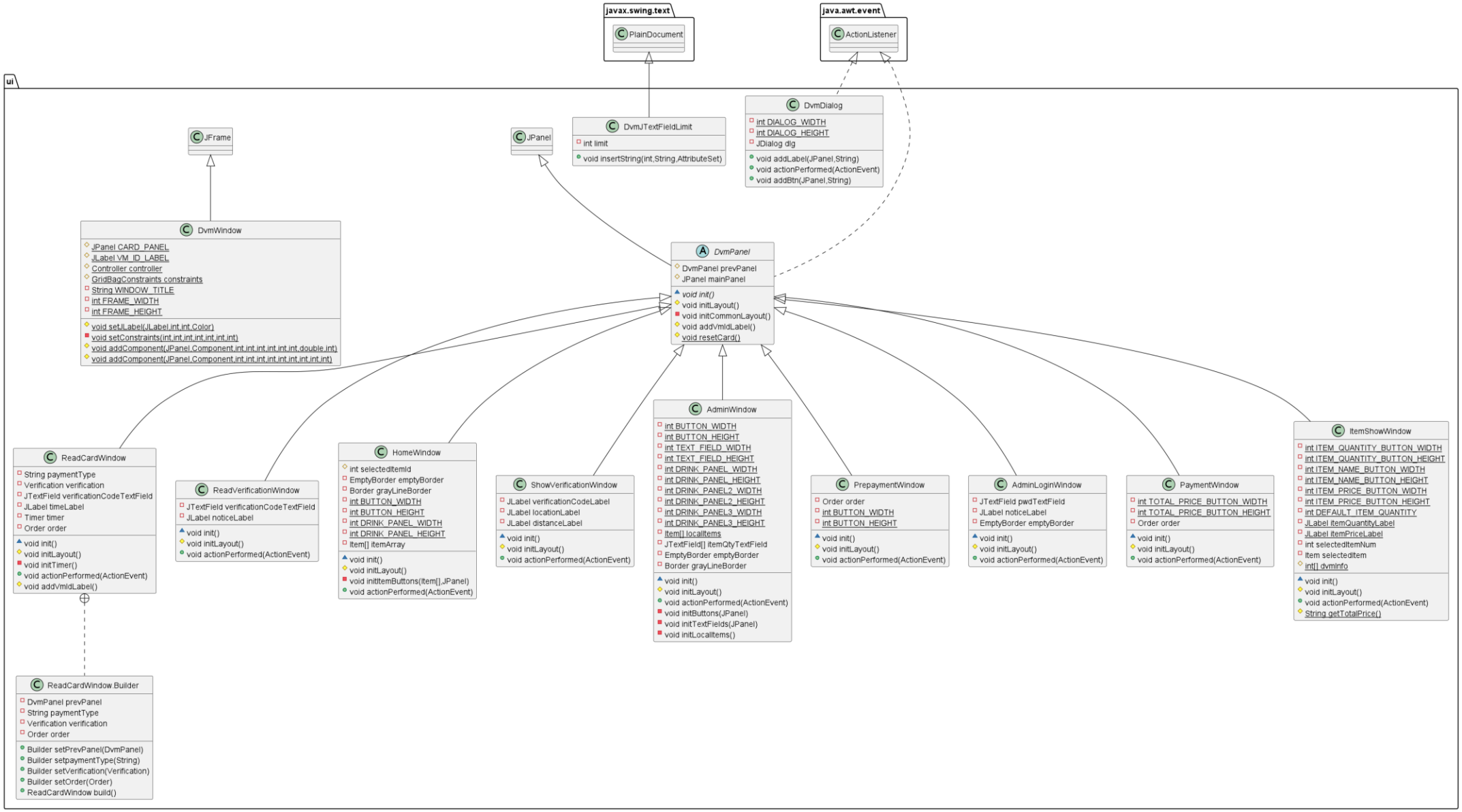




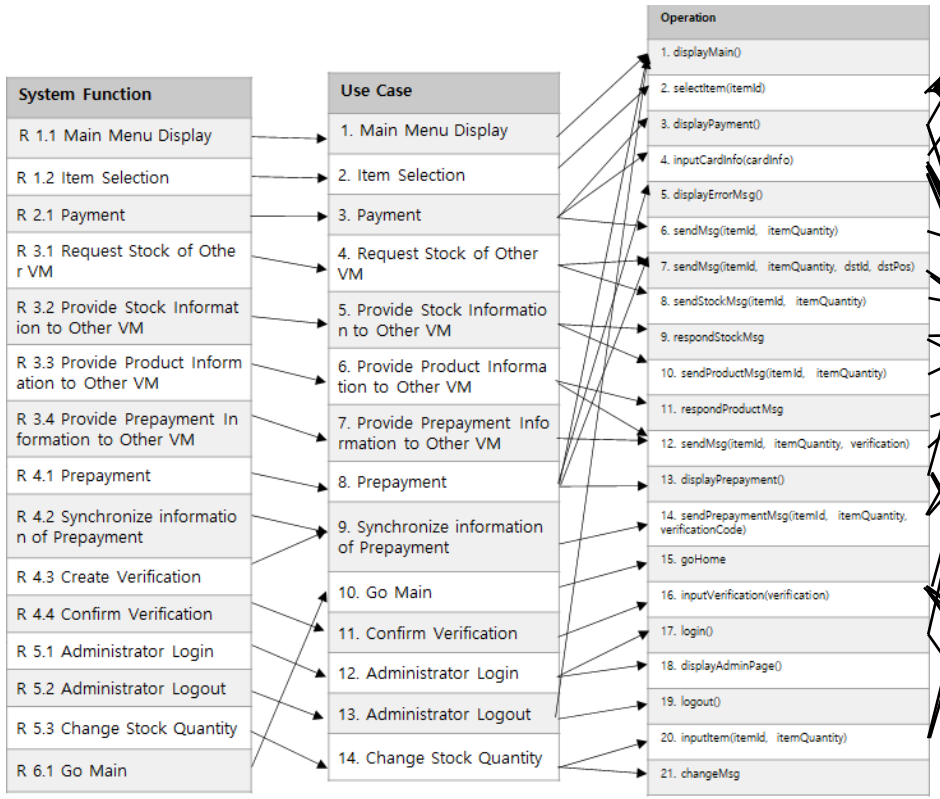
CLASS DIAGRAM



CLASS DIAGRAM



TRACEABILITY TABLE



METHOD	CLASS
getInstance(): Controller	Controller
selectItem(itemId:int, itemQuantity:int, dvmInfo:int[]):String	
payment(itemId:int, itemQuantity:int, cardNum:String, cardPwd:int): String	
prepayment(itemId:int, itemQuantity:int, cardNum:String, cardPwd:int, dstId:int): String	
getAccountManager(): AccountManager	
getItemManager():ItemManager	
getCardManager():CardManager	
getPaymentManager():PaymentManager	
getVerificationManager():VerificationManager	
getItemId(): int	Item
getItemQuantity():int	
setItemQuantity(itemQuantity:int):void	
getItemName():String	
getItemPrice():int	
getOnSale():boolean	
checkStock(itemId:int, itemQuantity:int):boolean	ItemManager
updateStockInfo(itemId:int. itemQuantity:int):void	
updateQuantity(itemId:int, itemQuantity:int):void	
checkProduct(itemId:int):boolean	
synchronize(itemId:int, itemQuantity:int, verificationCode:String):void	
getItemList():Item[]	
loadItemList():void	
checkStockOfOtherVM(itemId:int,itemQuantity:int):int[]	
sendMsg(dstId:int, msg:Message):void	MessageManager
sendPrepaymentInfo(itemId:int, itemQuantity:int, dstId:int, verificationCode:String):void	
sendStockMsg(itemId:int, itemQuantity:int, dstId:int):void	
sendProductMsg(itemId:int, itemQuantity:int, dstId:int):void	
setMsg(dstId:String, itemId:int, itemQuantity:int, msgType:String, authCode:String):Message	
setMsgDes(msgDes:Message.MessageDescription, itemId:int, itemQuantity:int, authCode:String):void	
decodeMsg(msg:message):int[]	
getNearestDvm():int[]	
getCardPwd():int	Card
getMoney():int	
setMoney(money:int):void	
checkCardValidity(cardNum:String, cardPwd:int):boolean	CardManager
getCardInfo(cardNum:String):Card	
encodeCardNum(inputCardNum:String):String	
loadCardList():void	
payment(cardManager:CardManager, totalPrice:int, cardNum:String):boolean	PaymentManager
cancelPayment(cardManager:CardManager, totalPrice:int, cardNum:String):void	
getVerificationValidity():boolean	Verification
getItemId(): int	
getItemQuantity():int	
getVerificationCode():String	
createVerificationCode():String	VerificationManager
saveVerification(itemId:int, itemQuantity:int, verificationCode:String, verificationValidity:boolean):void	
checkVerification(verificationCode:String):Verification	
isValidCode(code:String):boolean	
removeVerification(verificationCode:String):void	
verifyLoginInfo(password:String):boolean	AccountManager
getItem():Item	Order
getQuantity():int	

DESIGN PATTERN

Template Method Pattern (Before)

- Initiate Objects in every Window.
- Overlapping elements.

Ex) JFrame, JPanel, ...

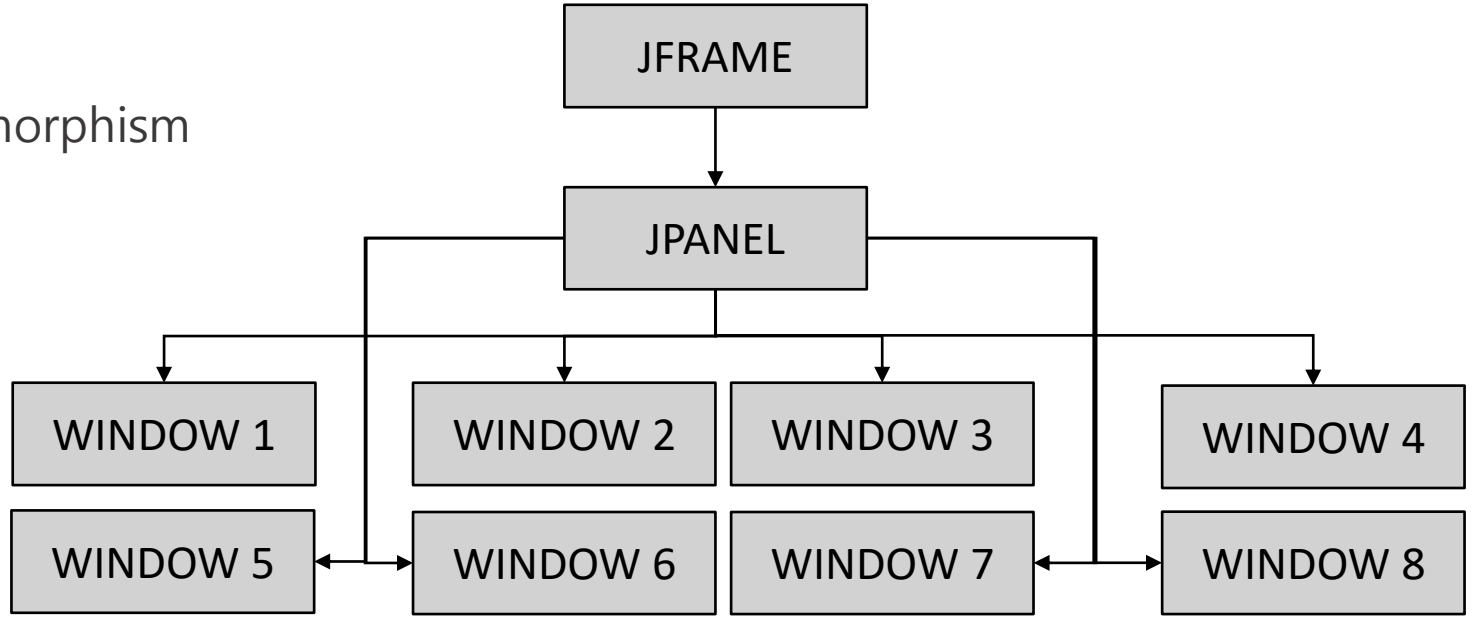
WINDOW 1	WINDOW 2	WINDOW 3	WINDOW 4
WINDOW 5	WINDOW 6	WINDOW 7	WINDOW 8

DESIGN PATTERN

Template Method Pattern (After)

- Reuse code from a Template

by Inheritance, Encapsulation, And Polymorphism



DESIGN PATTERN

Template Method Pattern (Before)

```
public class Window_1 extends JPanel implements ActionListener {
    public static int selectedItemId;
    private static final JButton btn1 = new JButton("ADMIN LOGIN");
    private static final JButton btn2 = new JButton("VERIFICATION CODE");

    private static final EmptyBorder eb = new EmptyBorder(new Insets(20, 10, 0, 10));
    private static final Border grayLine = BorderFactory.createLineBorder(Color.decode("#cfd0d1"), 1);

    private static final int btnWidth = 120;
    private static final int btnHeight = 30;
    private static final int drinkPanelWidth = 290;
    private static final int drinkPanelHeight = 400;

    public Window_1() {
        init();
    }

    protected void init() {
        JPanel panel = new JPanel(new GridBagLayout());
        JPanel itemLayout = new JPanel();

        itemLayout.setPreferredSize(new Dimension(drinkPanelWidth, drinkPanelHeight));
        itemLayout.setBorder(BorderFactory.createCompoundBorder(grayLine, eb));

        c = new GridBagConstraints();
        panel.setBackground(Color.decode("#dcebf7"));
        itemLayout.setBackground(Color.decode("#ebeced"));

        addJLabel(panel);
        addComponent(panel,btn1, 10, 2, 2, 10, 4, 0, 0.5, GridBagConstraints.FIRST_LINE_END);
        addComponent(panel, btn2, 10, 2, 2, 10, 4, 4, 0.5, GridBagConstraints.LINE_END);

        btn1.addActionListener(this);
        btn2.addActionListener(this);

        btn1.setFocusable(false);
        btn2.setFocusable(false);

        for (int i = 0; i < MAX_ITEM; i++) {
            JButton[] btn = new JButton[MAX_ITEM];
```

DESIGN PATTERN

Template Method Pattern (After)

```java
public abstract class DvmPanel extends JPanel implements ActionListener {
    protected DvmPanel prevPanel;
    protected JPanel mainPanel;

    public DvmPanel() {
        this(null);
    }

    public DvmPanel(DvmPanel prevPanel) {
        System.out.println(this.getClass() + "()");
        this.prevPanel = prevPanel;
    }

    abstract void init();

    protected void initLayout() {
        System.out.println(this.getClass() + ".initLayout()");
        initCommonLayout();
    }

    private void initCommonLayout() {
        mainPanel = new JPanel(new GridBagLayout());
        mainPanel.setBackground(Color.decode("#dcebf7"));
        constraints = new GridBagConstraints();
        CARD_PANEL.add(mainPanel);
        addVmIdLabel();
    }

    protected void addVmIdLabel() {
        constraints.insets = new Insets(2, 2, 2, 2);
        DvmWindow.VM_ID_LABEL.setBackground(Color.decode("#cfd0d1"));
        DvmWindow.VM_ID_LABEL.setOpaque(true);
        mainPanel.add(DvmWindow.VM_ID_LABEL, constraints);
    }
```

```java
// Window4
public class ProcessPaymentWindow extends DvmPanel {
    private String paymentType;

// ...

@Override
void init() {
    initLayout();
    initTimer();
}

@Override
protected void initLayout() {
    super.initLayout();

    verCode.setDocument(new JTextFieldLimit(CARD_NUM_LENGTH));
```

```java
// Window8
public class AdminWindow extends DvmPanel {

// ...

@Override
void init() {
    initLocalItems();
    initLayout();
}
```

DESIGN PATTERN

Builder Pattern (Before)

- Numerous constructors to create an object
- Leads to Telescoping Constructors Problem

```
public ReadCardWindow(DvmPanel prevPanel) {
    this(prevPanel, null, null);
}

public ReadCardWindow(DvmPanel prevPanel, String paymentType) {
    this(prevPanel, paymentType, null);
}

public ReadCardWindow(DvmPanel prevPanel, String paymentType, Verification verification) {
    super(prevPanel);
    this.paymentType = paymentType;
    this.verification = verification;
    System.out.println("Window4() with paymentType: " + paymentType + ", " + verification);
}
```

```
if (e.getActionCommand().equals("PAY")) {
    CARD.add(new ReadCardWindow(this, "prepayment"));
```

```
    new DvmDialog(resMsg);
    resetCard();
    CARD.add(new ReadCardWindow(this, "cancelPrepayment", verification));
}
```

DESIGN PATTERN

Builder Pattern (After)

- Creational Pattern
  (A way to build an object)
- Easy to modify according to attribute change
- Highly readable

```
private ReadCardWindow(DvmPanel prevPanel, String paymentType, Verification verification, Order order) {
    super(prevPanel);
    System.out.println("Window4() with paymentType: " + paymentType + ", " + verification);
    this.paymentType = paymentType;
    this.verification = verification;
    this.order = order;
    init();
}

public static class Builder {
    private DvmPanel prevPanel;
    private String paymentType;
    private Verification verification;
    private Order order;

    public Builder() {
        this.prevPanel = null;
        this.paymentType = null;
        this.verification = null;
        this.order = null;
    }

    public Builder setPrevPanel(DvmPanel prevPanel) {
        this.prevPanel = prevPanel;
        return this;
    }

    public Builder setpaymentType(String paymentType) {
        this.paymentType = paymentType;
        return this;
    }

    public Builder setVerification(Verification verification) {
        this.verification = verification;
        return this;
    }

    public Builder setOrder(Order order) {
        this.order = order;
        return this;
    }

    public ReadCardWindow build() {
        return new ReadCardWindow(prevPanel, paymentType, verification, order);
    }
}
```

```
if (e.getActionCommand().equals("PAY")) {
    CARD_PANEL.add(new ReadCardWindow.Builder()
            .setPrevPanel(prevPanel)
            .setpaymentType("prepayment")
            .setOrder(order)
            .build()
    );
```

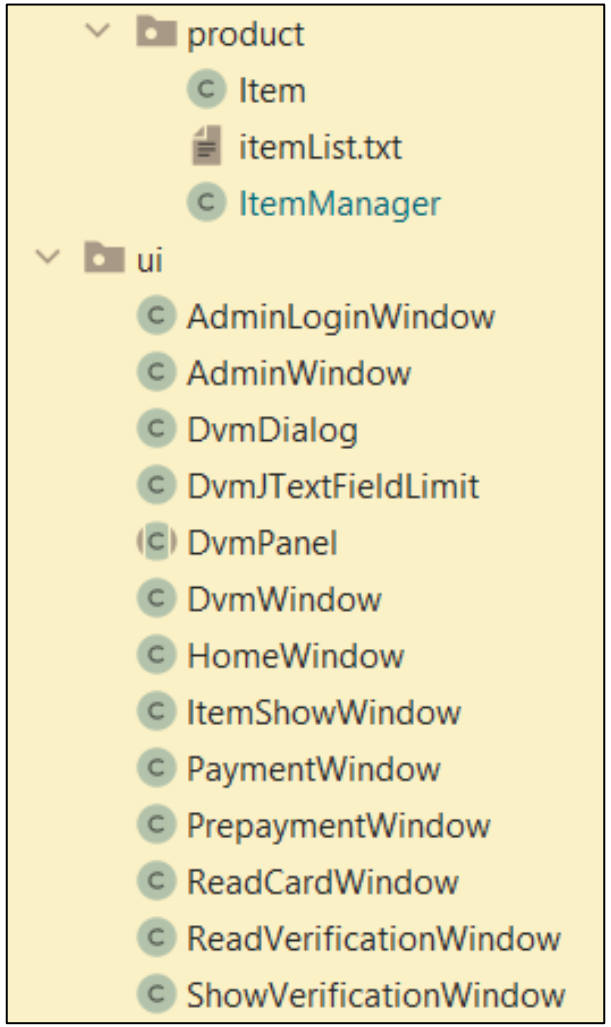
```
new DvmDialog(resMsg);
resetCard();
CARD_PANEL.add(new ReadCardWindow.Builder()
        .setPrevPanel(prevPanel)
        .setpaymentType("cancelPrepayment")
        .setVerification(verification)
        .build()
);
```

CLEAN CODE

<Clean Code>

1. Used Meaningful Names
   i. Variables
   ii. Class Name (Windows in UI)

2. Implemented Descriptive Functions.
   i. Standardized method
   ex) Init()




```
JButton loginBtn = new JButton( text: "LOGIN");
loginBtn.setFocusable(false);
loginBtn.addActionListener( l: this);
addComponent(mainPanel, loginBtn,  top: 0,  left: 0,

JButton backBtn = new JButton( text: "BACK");
backBtn.setFocusable(false);
backBtn.addActionListener( l: this);
addComponent(mainPanel, backBtn,  top: 10,  left: 2,
```

```
private static void setConstraints(int top,
    int left, int bottom, int right, int gridX, int gridY, int anchor) {
    constraints.insets = new Insets(top, left, bottom, right);
    constraints.gridx = gridX;
    constraints.gridy = gridY;
    constraints.anchor = anchor;
}

protected static void addComponent(JPanel panel, Component comp,
    int top, int left, int bottom, int right,
    int gridX, int gridY, double weightX, int anchor) {
    setConstraints(top, left, bottom, right, gridX, gridY, anchor);
    constraints.weightx = weightX;
    panel.add(comp, constraints);
}
```

WHY DO WE NEED OOAD(UP)?

ADVANTAGES:

- 시각적이고 정제된 문서화
- 프로젝트의 리스크 관리 용이
- 업무 분담 용이
- 프로젝트 설계의 유연성
- 사용자와의 피드백(커뮤니케이션) 원활

WHY DO WE NEED OOAD(UP)?

DISADVANTAGES:

- 과정이 복잡하다.
- 자칫하면 개발 방법론에 집착하기 쉽다.
- 설계/문서화에 지나치게 시간을 뺏길 수 있다.
- 리스크 관리에 의존적일 수 있다.
- 다소 주관적인 요소가 많음 -> 팀원과의 의견 마찰

WHY DO WE NEED OOAD(UP)?

적용 가능성 / 개선점

- 추상적 -> 개념 학습의 어려움
- 실제 적용 사례 조사 및 연구
- 프로젝트 경험이 부족한 경우

  -> 작은 규모부터 점차적으로 적용 학습

WHY DO WE NEED OOAD(UP)?

현업에서의 OOAD

- 도메인 전문가, 아키텍트, 개발자, 사용자

  OOAD(UP) 상에서 역할 분담에 대한 전반적인 이해.
  - ex) 용어, 과정, 원칙, 이점, 취지

- 개발 방법론 도입 대비 비용 계산
  - 개발 도구 사용 비용
  - 설계 비용, 코드 자동생성을 통한 개발 기간 단축/생산성 향상

- 초/중/고급 인력의 적절한 배치

WHY DO WE NEED OOAD(UP)?

느낀점

- 결국에는 더 나은 소프트웨어를 만들기 위한 목적
- 개별 프로젝트에 맞게 개발 프로세스(방법론) 조정
- 얼마나 OOAD의 장점을 기술적으로 살릴 수 있는가

감사합니다