Final Development Demonstration

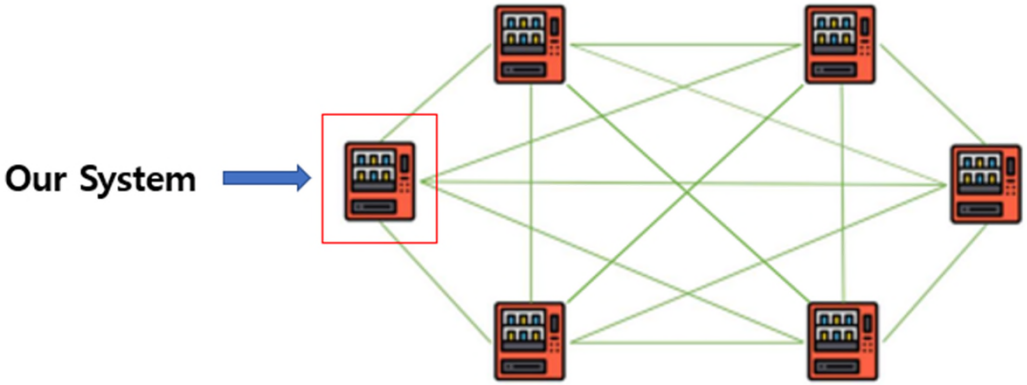
201612020 조석현 201811300 하승래
202010375 김만재 201811220 정유빈

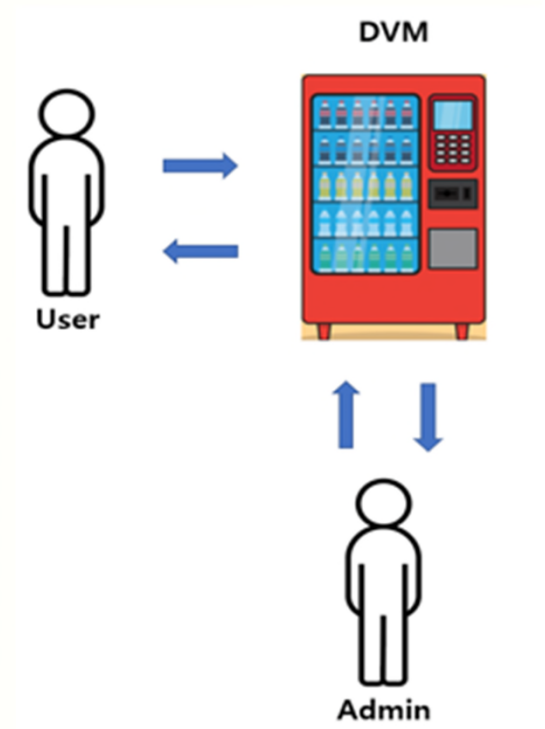
Contents

1. DVM 개발 과정

Ref.#	Function
R 1.1	구입할 상품 선택
R 1.2	인증 코드 입력
R 2.1	결제
R 2.2	상품 제공
R 2.3	타 자판기 재고 및 위치 안내
R 2.4	인증 코드 발급
R 3.1	재고 정보 송수신
R 3.2	판매 처리
R 4.1	관리자 메뉴 진입
R 4.2	재고 관리
R 4.3	위치 수정
R 4.4	판매 실적 합산

1. DVM 개발 과정

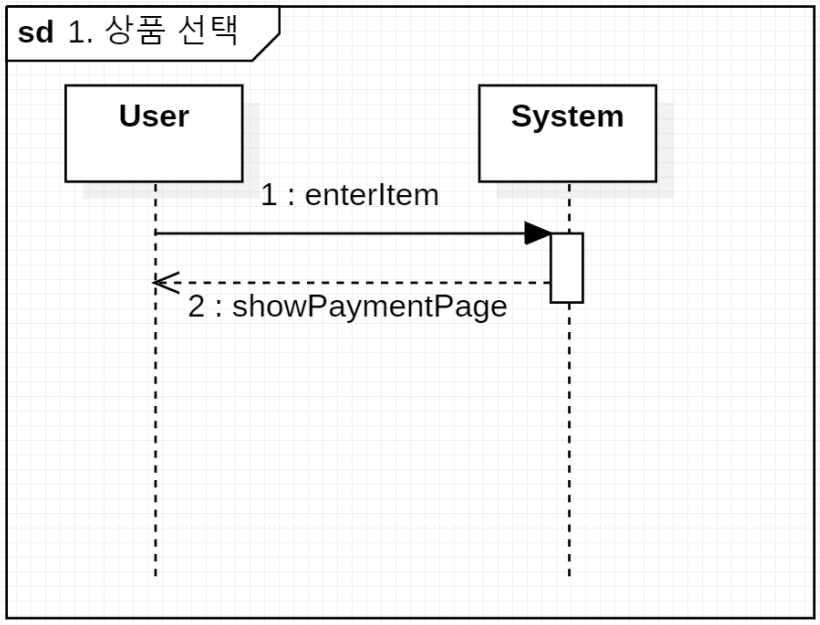


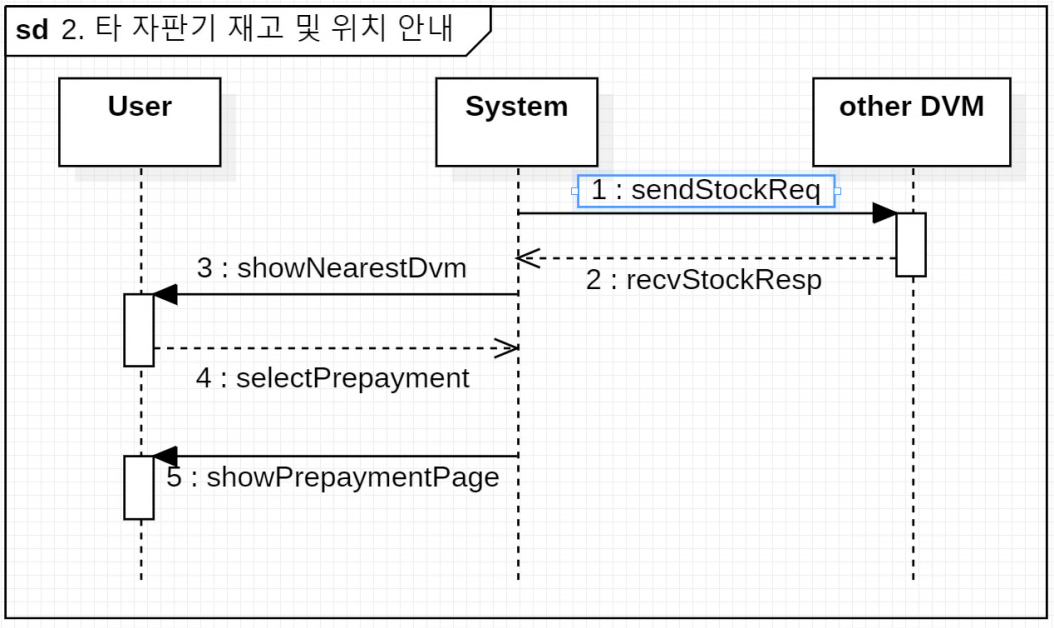


1. DVM 개발 과정

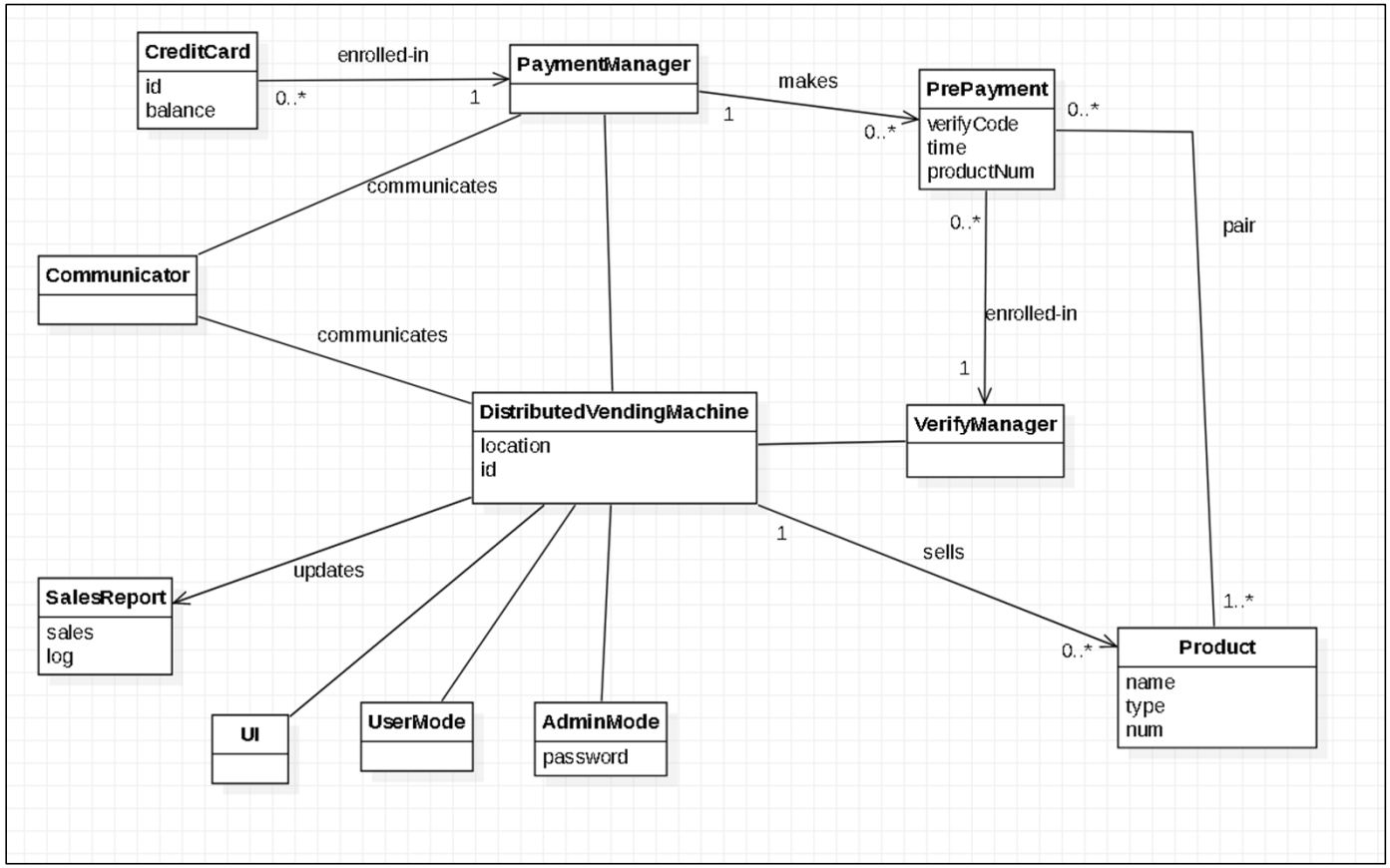
Ref.#	Function	Use Case Number & Name	Category
R 1.1	구입할 상품 선택	1. 상품 선택	Evident
R 1.2	인증 코드 입력	8. 인증 코드 입력	Evident
R 2.1	결제	3. 결제, 4. 선결제	Evident
R 2.2	상품 제공	3. 결제, 8. 인증코드 입력	Evident
R 2.3	타 자판기 재고 및 위치 안내	2. 타 자판기 재고 및 위치 안내	Evident
R 2.4	인증 코드 발급	5. 인증 코드 발급	Hidden
R 3.1	재고 정보 송수신	6. 재고 정보 송수신	Evident
R 3.2	판매 처리	7. 판매 처리	Evident
R 4.1	관리자 메뉴 진입	9. 관리자 메뉴	Evident
R 4.2	재고 관리	10. 재고 관리	Evident
R 4.3	위치 수정	11. 위치 수정	Evident
R 4.4	판매 실적 합산	12. 판매 실적 합산	Evident

1. DVM 개발 과정


sd 1. 상품 선택
User
System
1 : enterItem
2 : showPaymentPage

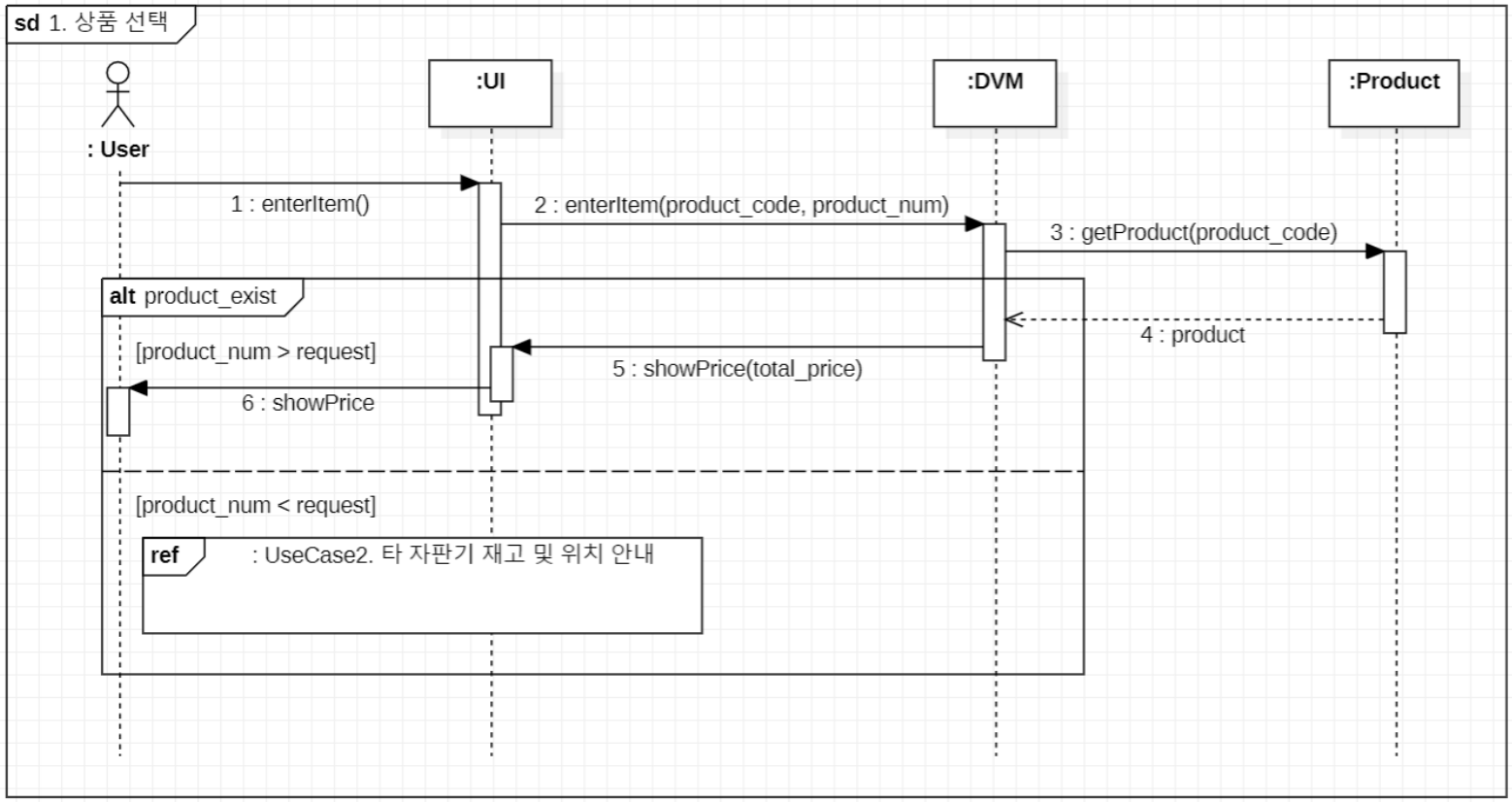

sd 2. 타 자판기 재고 및 위치 안내
User
System
other DVM
1 : sendStockReq
2 : recvStockResp
3 : showNearestDvm
4 : selectPrepayment
5 : showPrepaymentPage

1. DVM 개발 과정

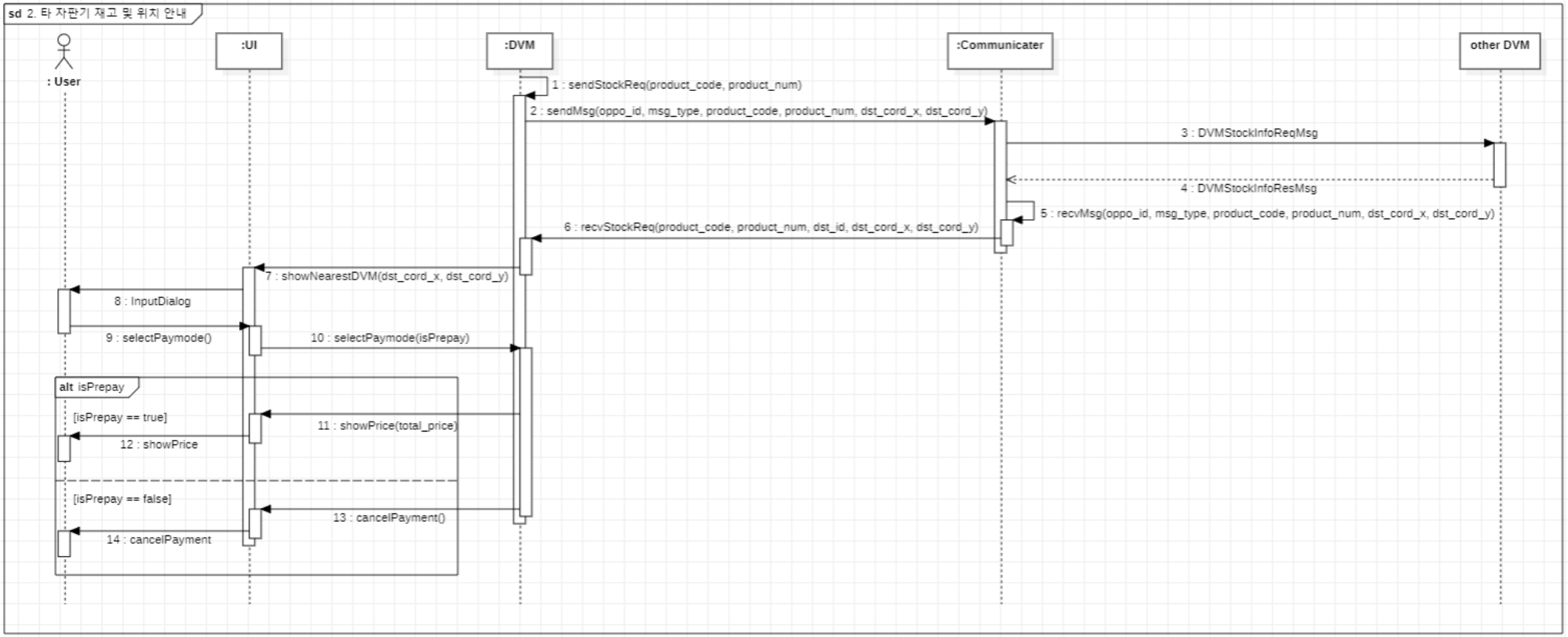


Operation In Sequence Diagram
enterItem
sendStockReq
showNearestDVM
selectPrepayment
showPrepaymentPage
showPrice
insertCard
DeployProduct
showVerificationPage
sendPrepayData
showVerificationCode
recvStockReq
broadcastStockData
sendStockData
enterVerificationCode
enterPw
showAdminPage
fillProduct
enterCord
reqSalesData
showSalesData

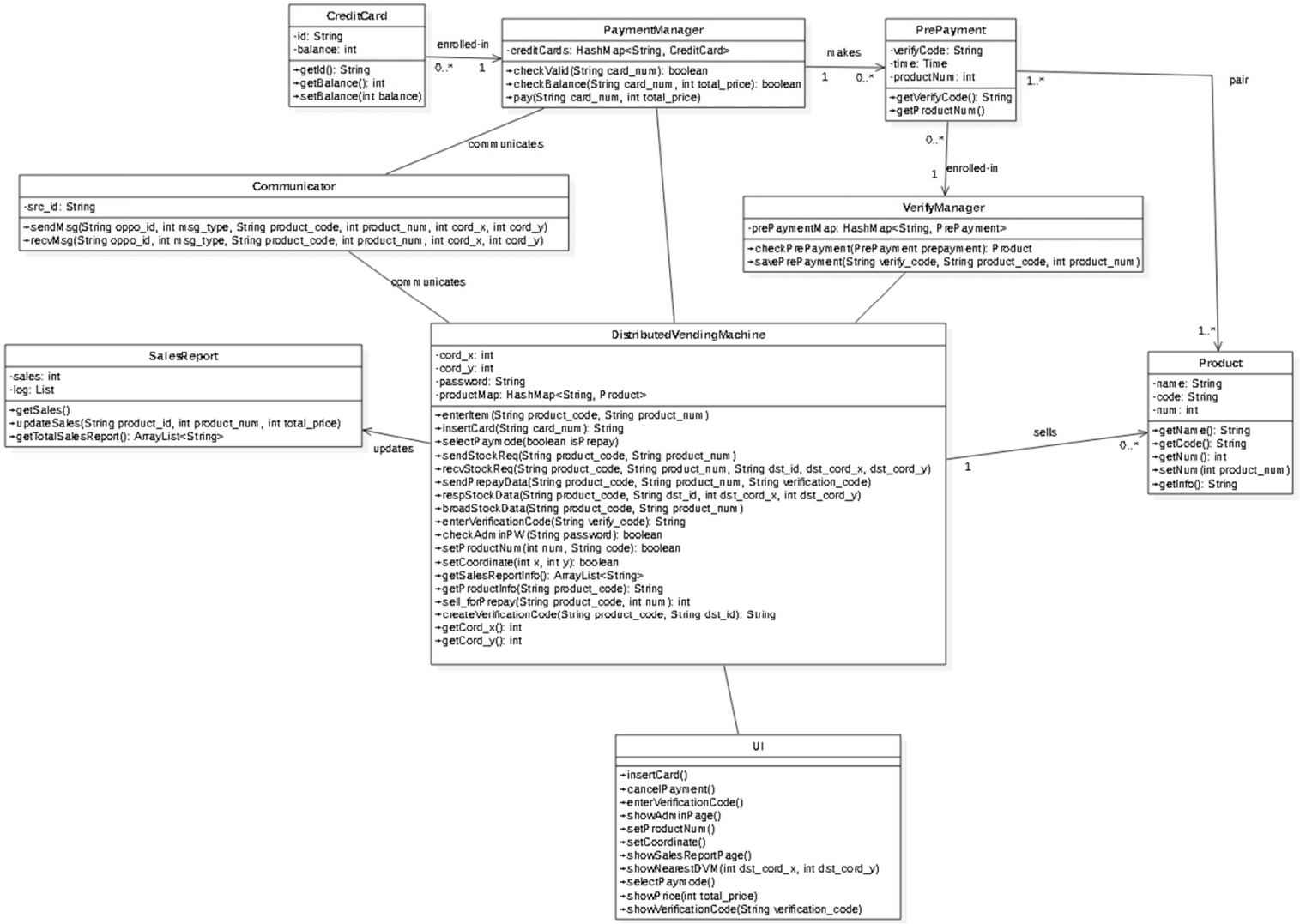
1. DVM 개발 과정


sd 1. 상품 선택
: User
:UI
:DVM
:Product
1 : enterItem()
2 : enterItem(product_code, product_num)
3 : getProduct(product_code)
4 : product
alt product_exist
[product_num > request]
5 : showPrice(total_price)
6 : showPrice
[product_num < request]
ref
: UseCase2. 타 자판기 재고 및 위치 안내

1. DVM 개발 과정


sd 2. 타 자판기 재고 및 위치 안내
: User
:UI
:DVM
:Communicater
other DVM
1 : sendStockReq(product_code, product_num)
2 : sendMsg(oppo_id, msg_type, product_code, product_num, dst_cord_x, dst_cord_y)
3 : DVMStockInfoReqMsg
4 : DVMStockInfoResMsg
5 : recvMsg(oppo_id, msg_type, product_code, product_num, dst_cord_x, dst_cord_y)
6 : recvStockReq(product_code, product_num, dst_id, dst_cord_x, dst_cord_y)
7 : showNearestDVM(dst_cord_x, dst_cord_y)
8 : InputDialog
9 : selectPaymode()
10 : selectPaymode(isPrepay)
alt isPrepay
[isPrepay == true]
11 : showPrice(total_price)
12 : showPrice
[isPrepay == false]
13 : cancelPayment()
14 : cancelPayment

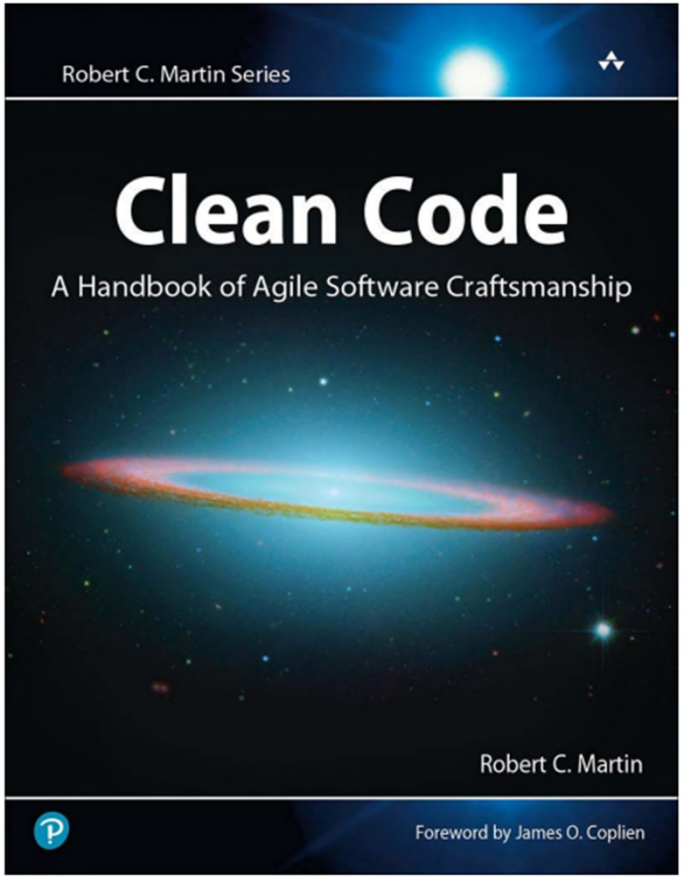
1. DVM 개발 과정


CreditCard
-id: String
-balance: int
+getId(): String
+getBalance(): int
+setBalance(int balance)
enrolled-in
0..*
1
PaymentManager
-creditCards: HashMap<String, CreditCard>
+checkValid(String card_num): boolean
+checkBalance(String card_num, int total_price): boolean
+pay(String card_num, int total_price)
makes
1
0..*
PrePayment
-verifyCode: String
-time: Time
-productNum: int
+getVerifyCode(): String
+getProductNum()
1..*
pair
communicates
0..*
1
enrolled-in
Communicator
-src_id: String
+sendMsg(String oppo_id, int msg_type, String product_code, int product_num, int cord_x, int cord_y)
+recvMsg(String oppo_id, int msg_type, String product_code, int product_num, int cord_x, int cord_y)
VerifyManager
-prePaymentMap: HashMap<String, PrePayment>
+checkPrePayment(PrePayment prepayment): Product
+savePrePayment(String verify_code, String product_code, int product_num)
communicates
1..*
SalesReport
-sales: int
-log: List
+getSales()
+updateSales(String product_id, int product_num, int total_price)
+getTotalSalesReport(): ArrayList<String>
updates
DistributedVendingMachine
-cord_x: int
-cord_y: int
-password: String
-productMap: HashMap<String, Product>
+enterItem(String product_code, String product_num)
+insertCard(String card_num): String
+selectPaymode(boolean isPrepay)
+sendStockReq(String product_code, String product_num)
+recvStockReq(String product_code, String product_num, String dst_id, dst_cord_x, dst_cord_y)
+sendPrepayData(String product_code, String product_num, String verification_code)
+respStockData(String product_code, String dst_id, int dst_cord_x, int dst_cord_y)
+broadStockData(String product_code, String product_num)
+enterVerificationCode(String verify_code): String
+checkAdminPW(String password): boolean
+setProductNum(int num, String code): boolean
+setCoordinate(int x, int y): boolean
+getSalesReportInfo(): ArrayList<String>
+getProductInfo(String product_code): String
+sell_forPrepay(String product_code, int num): int
+createVerificationCode(String product_code, String dst_id): String
+getCord_x(): int
+getCord_y(): int
sells
1
0..*
Product
-name: String
-code: String
-num: int
+getName(): String
+getCode(): String
+getNum(): int
+setNum(int product_num)
+getInfo(): String
UI
+insertCard()
+cancelPayment()
+enterVerificationCode()
+showAdminPage()
+setProductNum()
+setCoordinate()
+showSalesReportPage()
+showNearestDVM(int dst_cord_x, int dst_cord_y)
+selectPaymode()
+showPrice(int total_price)
+showVerificationCode(String verification_code)

1. DVM 개발 과정

1. DVM 개발 과정


A Brain-Friendly Guide
Head First
Design Patterns
Avoid those embarrassing coupling mistakes
Learn why everything your friends know about Factory pattern is probably wrong
Discover the secrets of the Patterns Guru
Load the patterns that matter straight into your brain
Find out how Starbuzz Coffee doubled their stock price with the Decorator pattern
See why Jim's love life improved when he cut down his inheritance
O'REILLY
Eric Freeman & Elisabeth Freeman
with Kathy Sierra & Bert Bates


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
Foreword by James O. Coplien

2. Traceability

SystemFunctions	Use Case	Operation In Sequence Diagram
R 1.1 구입할 상품 선택	1. 상품 선택	enterItem
R 1.2 인증 코드 입력	2. 타 자판기 재고 및 위치 안내	sendStockReq
R 2.1 결제	3. 결제	showNearestDVM
R 2.2 상품 제공	4. 선결제	selectPrepayment
R 2.3 타 자판기 재고 및 위치 안내	5. 인증 코드 발급	showPrepaymentPage
R 2.4 인증코드 발급	6. 재고 정보 송수신	showPrice
R 3.1 재고 정보 송수신	7. 판매 처리	insertCard
R 3.2 판매 처리	8. 인증 코드 입력	DeployProduct
R 4.1 관리자 메뉴 진입	9. 관리자 메뉴	showVerificationPage
R 4.2 재고 관리	10. 재고 관리	sendPrepayData
R 4.3 위치 수정	11. 위치 수정	showVerificationCode
R 4.4 판매 실적 합산	12. 판매 실적 합산	recvStockReq
		broadcastStockData
		sendStockData
		enterVerificationCode
		enterPw
		showAdminPage
		fillProduct
		enterCord
		reqSalesData
		showSalesData

2. Traceability

Operation In Sequence Diagram
enterItem
sendStockReq
showNearestDVM
selectPrepayment
showPrepaymentPage
showPrice
insertCard
DeployProduct
showVerificationPage
sendPrepayData
showVerificationCode
recvStockReq
broadcastStockData
sendStockData
enterVerificationCode
enterPw
showAdminPage
fillProduct
enterCord
reqSalesData
showSalesData

Operation in interaction diagram
enterItem( )
enterItem(product_code, product_num)
getProduct(product_code)
showPrice()
sendStockReq(product_code, product_num)
sendMsg(oppo_id, msg_type, product_code, product_num, dst_cord_x, dst_cord_y)
recvMsg(oppo_id, msg_type, product_code, product_num, dst_cord_x, dst_cord_y)
recvStockReq(product_code, product_num, dst_id, dst_cord_x, dst_cord_y)
showNearstDVM(dst_id)
selectPaymode()
selectPaymode(isPrepay)
showPrice(total_price)
insertCard( )
insertCard(card_num)
checkValid(card_num)
cancelPayment( )
checkBalance(card_num, total_price)
getBalance( )
pay(card_num, total_price)
setBalance(balance)
setNum(product_num)
updateSales(product_id, product_num, total_price)
createVerificationCode(product_code, dst_id)
sendPrepayData(product_code, product_num, verification_code)
respStockData(product_code, dst_id, dst_cord_x, dst_cord_y)
showVerificationCode(verification_code)
broadStockData(product_code, product_num)
getProductInfo(product_code)
getCoordinate( )
sell_forPrepay(product_code, num)
savePrePayment(verify_code, product, product_num)
enterVerificationCode( )
enterVerificationCode(verify_code)
checkPrePayment(verify_code)
showAdminPage( )
checkAdminPw(password)
setProductNum( )
setProductNum(num, code)
setCoordinate( )
setCoordinate(x, y)
showSalesReportPage( )
getSalesReportInfo( )
getTotalSalesReport( )

Method	Class
enterItem(String product_code, String product_num)	DistributedVendingMachine
showPrice()	
insertCard(String card_num)	
selectPrepayment()	
sendStockReq(String product_code, String product_num)	
recvStockReq(String product_code, String product_num, String dst_id, dst_cord_x, dst_cord_y)	
sendPrepayData(String product_code, String product_num, String verification_code)	
respStockData(product_code, dst_id, dst_cord_x, dst_cord_y)	
broadStockData(product_code, iproduct_num)	
enterVerificationCode(Prepayment)	
checkAdminPW(password)	
setProduct(num, code)	
setCoordinate(x, y)	
getSalesReport()	
getProduct(productCode)	
sell(product, num)	
createVerificationCode(product_code, dst_id)	
getName()	Product
getCode()	
getNum()	
setNum(product_num)	
checkValid(card_num)	PaymentManager
getBalance(card_num)	
pay(card_num, total_price)	
getId()	CreditCard
getBalance()	
setBalance(balance)	
getVerifyCode()	Prepayment
getProductNum()	
checkPrePayment(in PrePayment): Product	VerifyManager
savePrePayment(verifyCode, product, productNum)	
sendMsg(oppo_id, msg_type, msg_des, msg_dir)	Communicator
recvMsg(oppo_id, msg_type, msg_des, msg_dir)	
getSales()	SalesReport
updateSales(product_id, product_num, total_price)	
getTotalSalesReport()	

3. Design Pattern 적용

1) Template Method Pattern

```
public String insertCard(String card_num, String product_code, int num) {
    PaymentManager paymentManager = PaymentManager.getInstance();
    boolean valid = paymentManager.checkValid(card_num);
    if (valid == false) {

        return "유효하지않음";
    } // seq3 로직이 조금 이상한 것 같아요 의도에 맞는건지
    Product product = productMap.get(product_code);
    int total_price = product.getPrice() * num;
    boolean enough = paymentManager.checkBalance(card_num, total_price);
    if (enough == false) {
        return "잔액부족";
    }
    synchronized (this) {

        paymentManager.pay(card_num, total_price);
        product.setNum(product.getNum() - num);
    }
    Communicator.getInstance().broadStockData(product_code, num);
    SalesReport.getInstance().updateSales(product_code, num, total_price);

    StringBuilder sb = new StringBuilder();
    sb.append(product.getName()).append(' ').append(num).append("개 가져가세요.");

    return sb.toString();
```

3. Design Pattern 적용

1) Template Method Pattern

```
public String preinsertCard(String card_num, String product_code, int num) {
    PaymentManager paymentManager = PaymentManager.getInstance();
    boolean valid = paymentManager.checkValid(card_num);
    if (valid == false) {
        return "유효하지않음";
    } // seq3 로직이 조금 이상한 것 같아요 의도에 맞는건지
    Product product = productMap.get(product_code);
    int total_price = product.getPrice() * num;
    boolean enough = paymentManager.checkBalance(card_num, total_price);
    if (enough == false) {
        return "잔액부족";
    }
    paymentManager.pay(card_num, total_price);
    //선결제

    return "선결제성공";
}
```

3. Design Pattern 적용

1) Template Method Pattern

```java
public abstract class PaymentProcess {

    PaymentManager paymentManager;
    Product product;
    int totalPrice;
    SalesReport salesReport = new SalesReport();
    HashMap<String, Product> productMap;

    public final String paymentProcess(String cardNumber, String productCode, int num) {
        paymentManager = PaymentManager.getInstance();
        productMap = DistributedVendingMachine.makeExampleMap();

        insertCard(cardNumber, productCode, num);
        String result = pay(cardNumber, productCode, num, totalPrice);
        return result;
    }

    abstract String pay(String cardNumber, String productCode, int num, int totalPrice);

    String insertCard(String cardNumber, String productCode, int num) {
        boolean valid = paymentManager.checkValid(cardNumber);
        if (valid == false) {

            return "유효하지않음";    // 유효하지않음
        } // seq3 로직이 조금 이상한 것 같아요 의도에 맞는건지
        product = productMap.get(productCode);
        int totalPrice = product.getPrice() * num;
        boolean enough = paymentManager.checkBalance(cardNumber, totalPrice);
        if (enough == false) {
            return "잔액부족";      // 잔액부족
        }
         this.totalPrice = totalPrice;
         return "결제 성공";
    }

}
```

3. Design Pattern 적용

1) Template Method Pattern

```
public class NormalPayment extends PaymentProcess{

    Communicator communicator = Communicator.getInstance();
    @Override
    String pay(String cardNumber, String productCode, int num, int totalPrice) {
        synchronized (this) {

            paymentManager.pay(cardNumber, totalPrice);
            product.setNum(product.getNum() - num);
        }
        communicator.broadStockData(productCode, num);
        salesReport.updateSales(productCode, num, totalPrice);

        StringBuilder sb = new StringBuilder();
        sb.append(product.getName()).append(' ').append(num).append("개 가져가세요.");

        return sb.toString();
    }
}
```

3. Design Pattern 적용

1) Template Method Pattern

```java
public class PrePayment extends PaymentProcess{
    @Override
    String pay(String cardNumber, String productCode, int num, int totalPrice) {
        paymentManager.pay(cardNumber, totalPrice);
        //선결제

        return "선결제성공";
    }
}
```

3. Design Pattern 적용

1) Template Method Pattern

결제 / 선결제 기능에 대해 적용

기존 DistributedVendingMachine.java 내에 있던 결제 / 선결제 함수를 새로운 PaymentProces 라는 abstract class 로 분리, PaymentProcess 를 extends 하는 NormalPayment / Prepayment 클래스가 기존의 역할을 수행

장점1: 결제 / 선결제 함수 중 같은 기능을 하는 부분을 통일하여 가시성이 높아짐

장점2: 결제 관련 코드가 분리되어 해당 부분을 유지 보수하기 용이해짐

단점: 기존에 없던 클래스와 함수가 새로 생겨 파일이 늘어남. 작성 코드도 200줄 정도 증가.

3. Design Pattern 적용

2) Singleton Pattern

DistrbutedVendingMachine.java, Communicater.java 와 같이 하나만 존재하는 객체에 대해 적용

기존 DistributedVendingMachine.java 와 Communicater.java 는 객체를 최소한으로 호출하여 사용하였지만,
Singleton Pattern을 적용하여 하나의 객체만으로서 사용

장점1: 다른 클래스의 인스턴스들이 데이터를 공유하기 용이해짐

장점2: 고정된 메모리 영역을 얻기 때문에 메모리 낭비를 방지할 수 있음

단점1: Encapsulation을 중요시하는 객체지향개발방법론과 살짝 거리가 있음

단점2: 멀티 스레드 환경에서 동기화 처리를 하는 등의 주의가 필요함

4. Clean Code 적용

Clean Code 적용 전/후 및 평가

Clean Code 적용 전: 언더바 표기, 카멜 표기 등이 섞여 있었으며 약어 사용 등으로 인해 변수명의 의미가 드러나지 않는 코드들이 존재했다.

Clean Code 적용 후: 표기법을 카멜 표기로 통일하였으며, 변수명의 의미가 명확히 드러나도록 코드를 수정하였다.

평가: 두번째 Iteration을 진행하면서 코드를 다시 살펴봤을 때, 한번에 이해가 되지 않는 코드들이 있었다.
Clean Code 기법을 적용함으로써 이러한 문제점을 해결할 수 있었으며 제 3자가 보았을 때도 이해가 어렵지 않은 코드를 작성하였다.

4. Clean Code 적용

Clean Code 적용 사례 - Communicator Class

Clean Code 적용 전

```
static class Queue_using_ArrayList {
    정유빈
    static Message front(ArrayList<Message> list) {
        return list.size() != 0 ? list.get(0) : null;
    }

    정유빈
    static void pop(ArrayList<Message> list) {
        if (list.size() != 0)
            list.remove( index: 0);
    }

    정유빈
    static void push(ArrayList<Message> list, Message msg) {
        list.add(msg);
    }
}
```

Clean Code 적용 후

```
static class MessageQueue {
    Jay +1
    static Message front(ArrayList<Message> messageList) {
        return messageList.size() != 0 ? messageList.get(0) : null;
    }

    Jay +1
    static void pop(ArrayList<Message> messageList) {
        if (messageList.size() != 0)
            messageList.remove( index: 0);
    }

    Jay +1
    static void push(ArrayList<Message> messageList, Message message) {
        messageList.add(message);
    }
}
```

5. OOAD를 적용해서 개발을 하며 느낀 점

OOAD 개발 방법론의 장점

1. UP 가 제시하는 방법론을 모두 올바르게 수행했을 때 빈틈없는 설계가 가능할 것 같다.
2. 유지보수에 유리하고, OOPT의 경우 매 단계 traceability table을 활용하여 이전 단계와 비교하며 현 단계의 결과물을 점검할 수 있다.
3. 협업의 측면에서 팀원 각자가 맡은 파트들 사이의 상호작용 인터페이스를 명확히 정의하여 코드 통합 단계에서 문제를 줄일 수 있다.
4. OOA, OOD, OOI를 반복하는 과정에서 개발하는 서비스에 대해 명확하게 이해할 수 있다.
5. Analysis 와 Design 단계에서 많은 노력을 쏟기 때문에, 구현단계에서 코드를 작성하기 쉽다.

5. OOAD를 적용해서 개발을 하며 느낀 점

OOAD 개발 방법론의 단점

1. 코드를 작성하면서 나타날 수 있는 문제에 대해서 미리 대비하기 어렵다.

2. OOAD의 high-level 단계(OOA 단계) 에서 앞으로 사용하게 될 아키텍처와, 라이브러리 및 API의 인터페이스에 대해 어느정도 숙지하고 있어야 한다.

3. 인터페이스를 잘 모르고 sequence diagram을 설계하면 나중에 구현 단계에서 뒤늦게 깨닫고 다시 역으로 문서를 수정해야한다. (DVM_Network를 구현하는 과정에서 두드러지는 측면이 있음.)

4. UML tool (특히 Star UML)의 사용법이 어려우며 low-level에 가까운 설계일수록, notation에 대한 높은 이해가 필요하다.

5. 결과물로 도출되는 문서의 양이 기존 개발방식에 비해 많다.

5. OOAD를 적용해서 개발을 하며 느낀 점

OOAD 개발 방법론의 적용가능성

1. 고객이 원하는 서비스 뿐만 아니라, 개인 프로젝트를 진행할 때도 어떤 기능을 원하는지 정리하고 구현할 때, 적용할 수 있을 것 같다.

2. 객체 기반 '협업' 및 유지보수의 입장에서 봤을 때 충분히 적용 가능하다.
(문서작성에 소요되는 시간과 노동력보다 재사용성, 유지보수성의 측면이 더 크다.)

5. OOAD를 적용해서 개발을 하며 느낀 점

OOAD 개발 방법론의 개선점

1. 구현에 시간이 조금 더 배분이 되었다면 더 좋을 결과물이 나올 수 있었을 것 같다.

2. 도출되는 문서가 너무 많다.

3. high-level의 sequence diagram과, class diagram으로도 어렵지 않게 구현할 수 있을 것 같다.

선결조건 및 지원방법: 고객이 원하는 기능을 정확하게 파악해야 하고, 고객은 원하는 바를 정확하게 전달해야 한다.
팀원들은 되도록 회의를 많이 해서 혼동이 오지 않도록 해야 한다.

5. OOAD를 적용해서 개발을 하며 느낀 점

OOAD 개발 방법론의 선결조건 및 지원 방법

1. 고객이 원하는 기능을 정확하게 파악해야 하고, 고객은 원하는 바를 명확하게 전달해야 한다.
2. 팀원들과 원활한 의사소통이 이루어져야 한다.
3. 충분한 실력/경험이 있는 senior가 조언을 줄 수 있거나, 선임 개발자이거나, 팀장인 팀에 적용해야 적합하다.
4. 각 iteration/waterfall 기반으로 단계별 문서 작성 요령 및 설계에 대한 팀/팀장의 교육과 피드백이 중요하다.
5. 좋은 UML 협업 도구가 필요하다.

감사합니다