Software Modeling

# A NEW RELIABLE ATM

OOPT 3rd Cycle-System Test

201411140 권성완
201511247 김선정
201510436 허윤아
201510285 조수빈

# CONTENTS


01. System Test 대응

02. Static Report 대응

03. 소감

# 01. SYSTEM TEST 대응

## OOPT STAGE 1000.

- Use case Description 관련 [page 14]

| Use Case | 2. Withdraw |
|---|---|
| Actors | Customer |
| Description | -Customer can withdraw cash from account by using card or bankbook.<br>-Customer can only withdraw money in unit of 10000₩ and 50000₩ under balance.<br>-Customer has withdrawing limit at a day.<br>-Customer needs to choose the number of 50000₩.<br>-Customer needs to know password of an account.<br>-If customer successfully withdraws, ATM requests Offer to update server information.(Hidden) |

➔ Use case 'Withdraw'의 description에 Stage 1003에 명시되어 있지 않은 'limit at a day'라는 개념이 새로 등장합니다. Stage 1003에 내용을 추가하여 일관성을 유지해야 합니다.

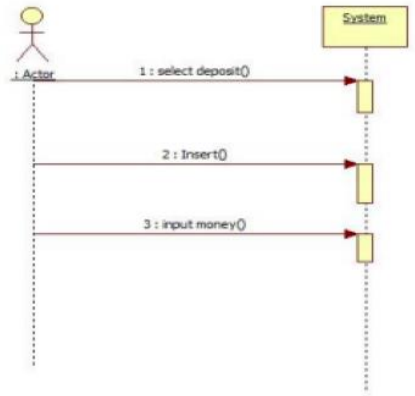
| Use Case | 3. Withdraw |
|---|---|
| Actors | Customer |
| Description | -Customer can withdraw cash from account.<br>-Customer can only withdraw money in unit of 10000₩, 50000₩ under balance.<br>-Customer needs to choose the number of 50000₩.<br>-Customer needs to know password of an account.<br>-If customer is withdrawing cash from other bank's account(not a default bank), ATM takes charge of certain percentage.<br>-If customer successfully withdraws, ATM requests Offer to update DB. |

-ATM에서 인출할 수 있는 현금을 무한대라고 가정,
-Withdraw 의 Description 중 limit a day라는 부분을 뺌.
-ATM에서 기본적으로 관리하는 은행의 계좌를 제외하고는 인출 시 수수료가 붙는다는 Description을 추가

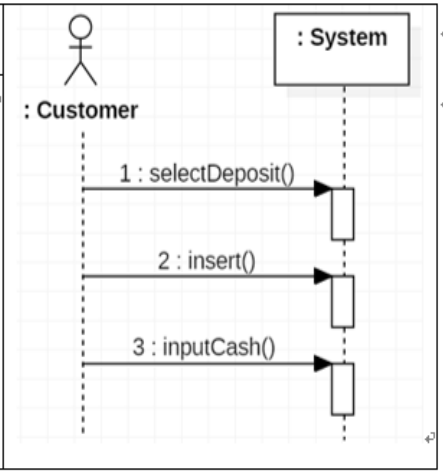
# 01. SYSTEM TEST 대응

## OOPT STAGE 2030.

Activity 2035. Define System Sequence Diagrams

| Use Case : Deposit | |
|---|---|
| 1. Customer selects deposit menu from the basic screen.<br>2. System prints out "Input check card or bankbook or credit card".<br>3. Customer inputs credit card, check card or bankbook.<br>4. If user information is valid(from "Insert" and "Request Customer's Data"), ask customer to input cash.<br>5. Customer Inputs cash in unit of 10000₩ and 50000₩.<br>6. System checks amount of cash inputted.<br>7. If total amount of money is correct, system invokes "Print transaction receipt".<br>8. (S) If Transaction Receipt is successfully printed, ask Offer to update server information. |  |

➔ 좌측 sequence에 명시된 8번 Transaction Receipt의 화살표가 존재하지 않습니다. 이에 따라 해당 단계에 대한 해석이 애매모호 해졌습니다.

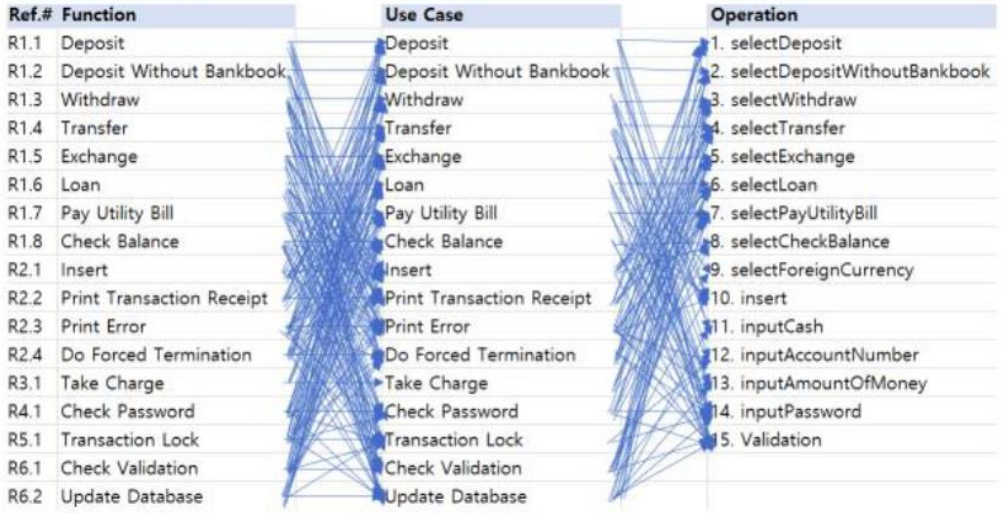
| Use Case: Deposit | |
|---|---|
| 1. Customer selects deposit menu from the basic screen.<br>2. Customer inputs account or credit card number.<br>3. Customer Inputs cash in unit of 10000₩, 50000₩.<br>4. System prints out transaction receipt. |  |

-기존의 너무 구체적이었던 Description을 간결하게 수정
-Print Transaction Receipt의 경우, sequence상 필요하다고 느껴져 빼지 않았지만, system operation이 아니므로 diagram에는 반영하지 않음.
-이 외의 다른 Use case들도 System sequence diagram과 일관성을 가지도록 description을 수정.

# 01. SYSTEM TEST 대응

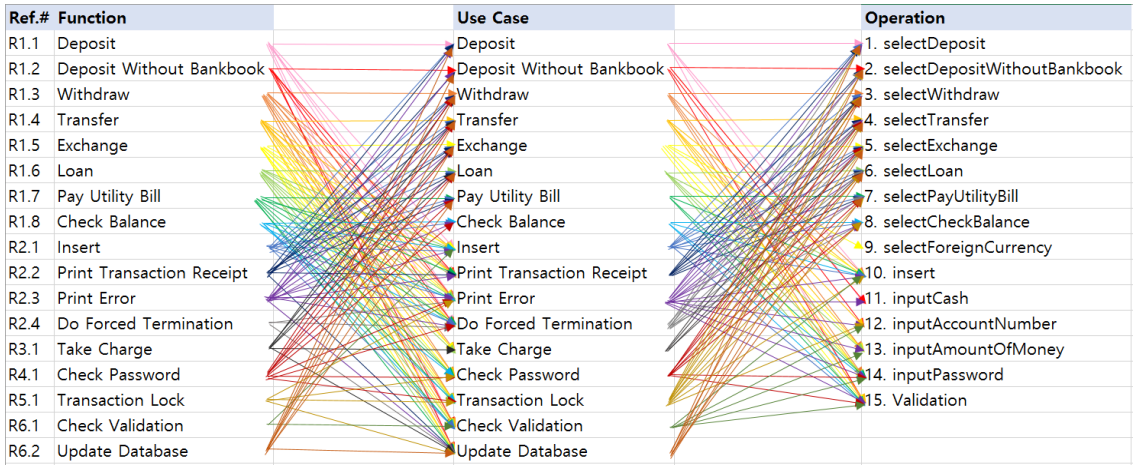
## OOPT STAGE 2030.

Activity 2039. Analyze (2030) Traceability Analysis



| Ref.# | Function | Use Case | Operation |
|---|---|---|---|
| R1.1 | Deposit | Deposit | 1. selectDeposit |
| R1.2 | Deposit Without Bankbook | Deposit Without Bankbook | 2. selectDepositWithoutBankbook |
| R1.3 | Withdraw | Withdraw | 3. selectWithdraw |
| R1.4 | Transfer | Transfer | 4. selectTransfer |
| R1.5 | Exchange | Exchange | 5. selectExchange |
| R1.6 | Loan | Loan | 6. selectLoan |
| R1.7 | Pay Utility Bill | Pay Utility Bill | 7. selectPayUtilityBill |
| R1.8 | Check Balance | Check Balance | 8. selectCheckBalance |
| R2.1 | Insert | Insert | 9. selectForeignCurrency |
| R2.2 | Print Transaction Receipt | Print Transaction Receipt | 10. insert |
| R2.3 | Print Error | Print Error | 11. inputCash |
| R2.4 | Do Forced Termination | Do Forced Termination | 12. inputAccountNumber |
| R3.1 | Take Charge | Take Charge | 13. inputAmountOfMoney |
| R4.1 | Check Password | Check Password | 14. inputPassword |
| R5.1 | Transaction Lock | Transaction Lock | 15. Validation |
| R6.1 | Check Validation | Check Validation | |
| R6.2 | Update Database | Update Database | |

➔ 모든 Function 과 Use case Operation의 관계들이 그려졌지만 정확한 관계가 보여지지 않는다. 추가적인 수정이 필요하다..



| Ref.# | Function | Use Case | Operation |
|---|---|---|---|
| R1.1 | Deposit | Deposit | 1. selectDeposit |
| R1.2 | Deposit Without Bankbook | Deposit Without Bankbook | 2. selectDepositWithoutBankbook |
| R1.3 | Withdraw | Withdraw | 3. selectWithdraw |
| R1.4 | Transfer | Transfer | 4. selectTransfer |
| R1.5 | Exchange | Exchange | 5. selectExchange |
| R1.6 | Loan | Loan | 6. selectLoan |
| R1.7 | Pay Utility Bill | Pay Utility Bill | 7. selectPayUtilityBill |
| R1.8 | Check Balance | Check Balance | 8. selectCheckBalance |
| R2.1 | Insert | Insert | 9. selectForeignCurrency |
| R2.2 | Print Transaction Receipt | Print Transaction Receipt | 10. insert |
| R2.3 | Print Error | Print Error | 11. inputCash |
| R2.4 | Do Forced Termination | Do Forced Termination | 12. inputAccountNumber |
| R3.1 | Take Charge | Take Charge | 13. inputAmountOfMoney |
| R4.1 | Check Password | Check Password | 14. inputPassword |
| R5.1 | Transaction Lock | Transaction Lock | 15. Validation |
| R6.1 | Check Validation | Check Validation | |
| R6.2 | Update Database | Update Database | |

서로 다른 색상을 이용하여 보기 쉽게 수정

# 01. SYSTEM TEST 대응

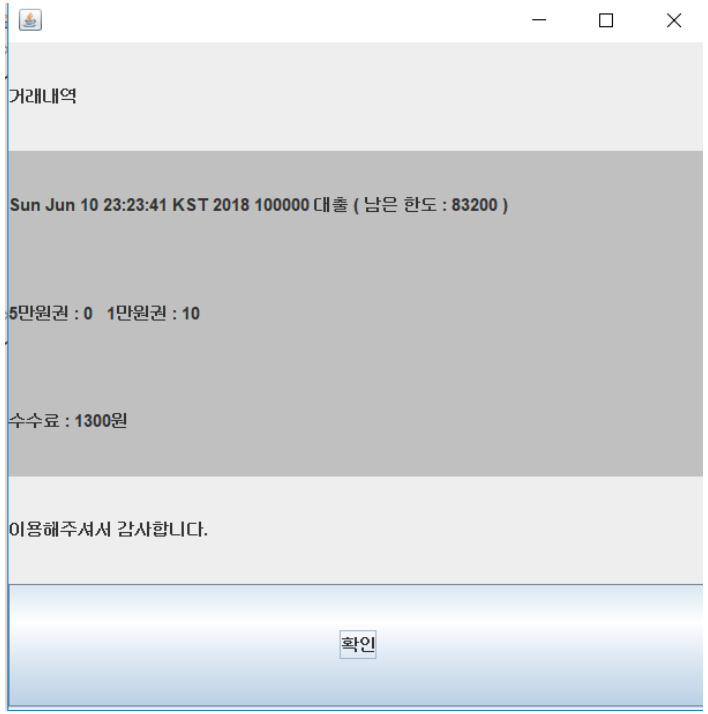
Brute Force Test.

| Test Case No. | 2 |
|---|---|
| 문제 | 무통장 입금시, 현대카드로 카드사 변경이 되지 않는다. |
| 대응 | 신한은행 ATM기로 지정해놨으므로 카드사 변경이 불가능하도록 설계되었으므로 수정이 불필요. |

| Test Case No. | 14 |
|---|---|
| 문제 | 단위가 맞지 않는다는 안내가 나오지 않는다. |
| 대응 | 안내가 나오도록 수정. |

# 01. SYSTEM TEST 대응

| Test Case No. | 17 |
|---|---|
| 문제 | 수수료가 처리되지 않는다. |
| 대응 | 출금, 대출 시 수수료가 처리되도록 수정 |




| Test Case No. | 18 |
|---|---|
| 문제 | 신용 등급이 제공되지 않는다. |
| 대응 | 대출 거래 진행시 신용 등급이 제공되도록 수정 |

# 01. SYSTEM TEST 대응

| Test Case No. | 20 |
|---|---|
| 문제 | 1만,10만,100만이 잘 표시되지 않는다. |
| 대응 | 환전에서 1만, 10만, 100만 버튼이 제대로 작동되도록 수정 |

Category Patitioning Test

| Test Case No. | 10 |
|---|---|
| 문제 | 존재하지 않는 계좌 번호를 입력해도 진행이 된다. |
| 대응 | 존재하지 않는 계좌 번호는 거래가 되지 않도록 수정. |

| Test Case No. | 14,18,22,26,38,42,46,58 |
|---|---|
| 문제 | 입금거래를 진행하며 0~10000원 사이의 금액을 입력할 수 없다. |
| 대응 | 시스템상에서 10000원 단위로만 입출금이 가능하도록 설정해놨기 때문에 수정의 필요성이 없다. |

| Test Case No. | 17,18,19,20,41,42,43,44,49,50,51,52,57,58,59,60 |
|---|---|
| 문제 | 거래가 정지된 계좌번호일 시에 금액, 계좌 입력이 되지 않는다. |
| 대응 | 거래가 정지된 계좌번호는 다음 단계로 진행할 수 없도록 설계했으므로 수정의 필요성이 없음. |

# 01. SYSTEM TEST 대응

| Test Case No. | 21,23,24,25 |
|---|---|
| 문제 | 무통장 입금 거래 진행시 정지되지 않은 계좌번호 입력과 적절한 금액 입력이 되지않는다. |
| 대응 | 무통장 입금이 되도록 코드 수정 |

| Test Case No. | 39,40,41 |
|---|---|
| 문제 | 거래가 정지되지 않은 계좌로 대출이 되지 않는다. |
| 대응 | 대출이 가능하도록 코드 수정. |

| Test Case No. | 45,47,48 |
|---|---|
| 문제 | 송금 거래 진행이 되지 않는다. |
| 대응 | 송금이 가능하도록 코드 수정 |

주요기능들의 미완성으로 인한 FAIL을 모두 수정

## 코드 중복 수정 전 : 모든 function method마다 중복해서 list_bank를 초기화해주었다.

```
// 체크카드 0, 통장 1, 신용카드 2, 지로용지3 // 은행 // 계좌정보
public void insert(String str) {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "삼성카드", "현대카드", "롯데카드" };
    this.card = str;
    this.account.setBank(list_bank[Integer.parseInt(card.charAt(0) + "")]);
    this.account.setAccountNumber(card.substring(1));
}

public int lockAccount() {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "삼성카드", "현대카드", "롯데카드" };
    int bank;
    for (bank = 0; bank < list_bank.length; bank++) {
        if (list_bank[bank].equals(account.getBank()))
            break;
    }
    if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
        account.setIsLocked(true);
        if (offer[bank].updateDatabase(account)) {
            return 0;
        } else
            return 2;
    } else
        return 3;
}

public int deposit(String money) {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "삼성카드", "현대카드", "롯데카드" };
    int bank;
```

# **코드 중복** 수정 후 : 변수와 method로 추가하여 가독성, 유지보수성 상승.

```
// String[] list_bank: 메소드마다 들어가 있던거 global로 뺌.
String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "삼성카드", "현대카드", "롯데카드" };

   private int findWhichBank(Account account) {
       // 어떤 은행/카드사인지 찾고 은행종류를 int로 리턴.
       int bank;
       for (bank = 0; bank < list_bank.length; bank++) {
           if (list_bank[bank].equals(account.getBank()))
               break;
       }
       return bank;
   }
```

findWhichBank 메소드 추가

적용화면

```
public int lockAccount() { // account를 lock하는 메소드.

    // 어느 은행인지 찾는다.
    int bank;
    bank = findWhichBank(account);
```

**If-else 정리** 전: : 모든 메소드마다 if(){여러코드내용} else~~

```
public int deposit(String money) {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은
    int bank;
    for (bank = 0; bank < list_bank.length; bank++) {
        if (list_bank[bank].equals(account.getBank()))
            break;
    }
    if (this.account.getBank().substring(2, 4).equals("카드")) {
        if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
            if (isBig(money, account.getDept())) {
                return 6;
            } else {
                account.setDept(minus(account.getDept(), money));
                String newLog = new Date() + " " + money + " 상환 \t( 대출금 : " +
                newLog = newLog + account.getLog();
                account.setLog(newLog);
                if (offer[bank].updateDatabase(account)) {
                    return 0;
                } else {
                    return 2;
                }
            }
        } else {
            return 3;
        }
    } else {
```

If 문 {} 안에 긴 내용이 들어가 있고,
코드가 끝나가는 마지막 부분엔 else가
반복되는 구조.
MainSystem 클래스의
모든 메소드의 구조가 동일했다.

**If-else 정리** 후:

Method추가 /조건식을 반대로 수정/ 조기 return문 이용.

```
public int deposit(String money) { // 입금 기능. return하는 에러들은 GUI에서 frame만들 때 쓰인다.

    int bank;
    bank = findWhichBank(account);
    if (isWrongAccount(bank, account))
        return WRONG_ACCOUNT;

    // 입력한 계좌가 카드번호인경우 빛 상환.
    if (this.account.getBank().substring(2, 4).equals("카드")) {
        return deposit_CARD(money, this.account, bank);
    } else { // 입력한 계좌가 은행 계좌일 경우.
        return deposit_BANK(money, this.account, bank);
    }
}
```

Else return 3; 없어짐

리턴을 빨리 해 주어서 이전처럼 if문 안에 주렁주렁 매달려 있는 코드 사라짐

+ else문의 반복도 사라짐

**Magic number** 정리 전: 리턴하는 숫자들이 무엇을 의미하는지 알 수 없음.

```
                newLog = newLog + account.getLog();
                account.setLog(newLog);
                if (offer[bank].updateDatabase(account)) {
                    return 0;
                } else {
                    newAccount.setBalance(plus(newAccount.getBalance(
                    tempLog = new Date() + " " + money + " 기존거래 취소 "
                            + account.getAccountNumber() + "\t( 잔액 :
                    tempLog = tempLog + newAccount.getLog();
                    newAccount.setLog(tempLog);
                    if (!offer[temp_bank].updateDatabase(newAccount))
                        return 2;
                    }
                    return 2;
                }
            }
        } else {
            return 3;
        }
    }
    return 3;
}
```

언제 3을 리턴하는지, 언제 2를 리턴하는지, 언제 0을 리턴하는지 알 수 없음.

각 숫자가 의미하는 뜻도 알 수 없음.

## 02. STATIC REPORT 대응

**Magic number** 정리 후 : 리턴되는 조건을 보다 쉽게 알 수 있음.

```
// 송신자에서 차감.
senderUpdateCheck = withdraw_Money(money, offer[bank], account, "송금 ", receiveAccount);

if (receiverUpdateCheck == SERVER_NOT_RESPONSE || senderUpdateCheck == SERVER_NOT_RESPONSE)
    // updateDB시 오류가 난 경우 거래 취소.

    withdraw_Money(money, offer[receiver_bank], receiveAccount, "기존 거래 취소");
    withdraw_Money(money, offer[bank], account, "기존 거래 취소");

    return SERVER_NOT_RESPONSE;
}
return NO_ERROR;
}
```

```
// receiver의 bank 얻어오고 valid하지 않으면 오류.
int receiver_bank;
receiver_bank = findWhichBank(receiveAccount);
if (isWrongAccount(receiver_bank, receiveAccount))
    return WRONG_ACCOUNT;
offer[receiver_bank].readDatabase(receiveAccount);

// 송금계좌의 잔액 확인
if (isUnderBalance(money))
    return NOT_ENOUGH_BALANCE;
```

리턴하던 숫자들을 상수로 만들어줘서 가독성과 유지보수성을 높였다

**If-else**문 정리 전 : 필요 이상으로 중첩된 if else문을 볼 수 있음

```
1  if (offer[bank].checkValid(account)) {
       if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
           int temp_bank = 20;
           for (int i = 0; i < 10; i++) {
               if (newAccount.getBank().equals(list_bank[i])) {
                   temp_bank = i;
                   break;
               }
           }
           if (temp_bank == 20) {
               return 2;
           }
2          if ((account.getBank().equals("신한은행") && isBig(money, account.getBalance()))
                   || (!account.getBank().equals("신한은행") && isBig(plus(money, "1300"), account.getBalance()))) {
               return 1;
           } else {
3              if (newAccount.getBank().substring(2, 4).equals("카드")) {
4                  if ((this.errorType = offer[bank].readDatabase(newAccount)) == 0) {
5                      if (isBig(money, newAccount.getDept())) {
                           return 6;
                       } else {
                           newAccount.setDept(minus(newAccount.getDept(), money));
                           String newLog = new Date() + " " + money + " 상환 \t( 대출금 : " + newAccount.getDept()
                                   + " )\n";
                           newLog = newLog + newAccount.getLog();
                           newAccount.setLog(newLog);
6                          if (!offer[bank].updateDatabase(newAccount)) {
                               return 2;
                           }
```

If(~){1
if(~){2
if(~){3
if(~){4....

최대 7중첩

**If-else문** 정리 후 : 중첩이 줄어들음.

```
// 송금하려는 account가 valid하지 않으면 오류.
if (!offer[bank].checkValid(account)) return WRONG_ACCOUNT;
if (isWrongAccount(bank, account)) return WRONG_ACCOUNT;
offer[bank].readDatabase(account);

// receiver의 bank 얻어오고 valid하지 않으면 오류.
int receiver_bank;
receiver_bank = findWhichBank(receiveAccount);
if (isWrongAccount(receiver_bank, receiveAccount)) return WRONG_ACCOUNT;
offer[receiver_bank].readDatabase(receiveAccount);

// 송금계좌의 잔액 확인
if (isUnderBalance(money)) return NOT_ENOUGH_BALANCE;

1 if (receiveAccount.getBank().substring(2, 4).equals("카드")) { // 수신자가 카드인 경우: 수신
      receiverUpdateCheck = deposit_CARD(money, receiveAccount, receiver_bank);
      // 송금자 잔액 차감
      senderUpdateCheck = withdraw_Money(money, offer[bank], account, "송금 ", receive
2     if (receiverUpdateCheck == 2 || senderUpdateCheck == 2) {// updateDB시 오류가 난 경

          // 수신자 거래 취소
          receiveAccount.setDept(plus(receiveAccount.getDept(), money));
          String newLog = new Date() + " " + money + " 기존거래 취소 \t( 대출금 : " + recei
          newLog = newLog + receiveAccount.getLog();
          receiveAccount.setLog(newLog);
```

최대 7중첩 -> 2중첩

# 02. STATIC REPORT 대응

규칙 위반

| 규칙 | 심각도 | 진단 메시지 |
|---|---|---|
| JAVA_66 | 매우 높음 | condition에 직접적인 assignment operator를 사용함. |

```
                break;
            }
            if (this.account.getBalance() < (won = money * exchangeRate[i])) {
                return 1;
```

```
            break;
        }
        if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
            if ((account.getBank().equals("신한은행") && isBig(money, account.getBalance()))
                    || (!account.getBank().equals("신한은행") && isBig(plus(money, "1300"), account.getBal
                return 1;
            } else {
                String newLog;
                if (!account.getBank().equals("신한은행")) {
                    this.takeCharge(account);
                }
                account.setBalance(minus(account.getBalance(), money));
                newLog = new Date() + " " + money + " 출금\t( 잔액 : " + account.getBalance() + " )\n" + a
                account.setLog(newLog);
```

## 02. STATIC REPORT 대응

규칙 위반

```
            break;
        }
        if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
            if ((account.getBank().equals("신한은행") && isBig(money, account.getBalance()))
                    || (!account.getBank().equ          은행") && isBig(plus(money, "1300"), account.getBal
                return 1;
            } else {
```

Method로 따로 빼줌 + 규칙 위반 정리
'(A=B)==C' 구조도 정리

```
}

public boolean isWrongAccount(int bankIndex, Account account) { // 잘못된 계좌: true 리턴.
    this.errorType = offer[bankIndex].readDatabase(account);
    boolean readWrongDB = (this.errorType != 0);
    return readWrongDB;
}
```

## 규칙 위반

| 규칙 | 심각도 | 진단 메시지 |
|---|---|---|
| JAVA_71 | 높음 | 클래스의 설명 주석이 없거나, 설명 주석 내용에 모든 태그가 포함되어 있지 않음 |

## 주석이 없음

```
public int lockAccount() {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "
    int bank;
    for (bank = 0; bank < list_bank.length; bank++) {
        if (list_bank[bank].equals(account.getBank()))
            break;
    }
    if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
        account.setIsLocked(true);
        if (offer[bank].updateDatabase(account)) {
            return 0;
        } else
            return 2;
    } else
        return 3;
}

public int deposit(String money) {
    String[] list_bank = { "국민은행", "기업은행", "농협은행", "신한은행", "씨티은행", "우리은행", "한국은행", "
    int bank;
    for (bank = 0; bank < list_bank.length; bank++) {
        if (list_bank[bank].equals(account.getBank()))
            break;
    }
    if (this.account.getBank().substring(2, 4).equals("카드")) {
        if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
            if (isBig(money, account.getDept())) {
                return 6;
            }
        }
    }
    if ((this.errorType = offer[bank].readDatabase(account)) == 0) {
        if ((account.getBank().equals("신한은행") && isBig(money, account.getB
                || (!account.getBank().equals("신한은행") && isBig(plus(money,
            return 1;
        } else {
            String newLog;
            if (!account.getBank().equals("신한은행")) {
                this.takeCharge(account);
            }
            account.setBalance(minus(account.getBalance(), money));
            newLog = new Date() + " " + money + " 출금 \t( 잔액 : " + account.g
            account.setLog(newLog);

            if (offer[bank].updateDatabase(account)) {
                return 0;
            } else {
                return 2;
            }
        }
    } else {
        return 3;
    }
```

규칙 위반

# 주석 추가 & 정리

```
public int lockAccount() { // account를 lock하는 메소드.

    // 어느 은행인지 찾는다.
    int bank;
    bank = findWhichBank(account);

    // 계좌가 비정상적인 계좌인지 확인.
    if (isWrongAccount(bank, account))
        return WRONG_ACCOUNT;

    // lock을 건다.
    account.setIsLocked(true);

    // DB에 update
    return checkUpdateDB(offer[bank], account);

}

public int deposit(String money) { // 입금 기능. return하는 에러
```

```
public int withdraw(String money) { // 출금한다.

    // 지역변수 bank. 어디 은행인지 알기 위함.
    int bank;
    bank = findWhichBank(account);

    // account 확인.
    if (isWrongAccount(bank, account))
        return WRONG_ACCOUNT;

    // 잔액 충분한지 확인
    if (isUnderBalance(money))
        return NOT_ENOUGH_BALANCE;
    else {
        return withdraw_Money(money, offer[bank], account, "출금");
    }
}

private int withdraw_Money(String money, Offer offer, Account acco
    // 돈 출금기능(수수료까지 출금o) + log업데이트 + checkUpdateDB
```

# EPILOGUE

-많은 Stage와 보고서량으로 인한 피로도가 컸다.
-이전의 보고서를 계속해서 다시 수정해야 했으므로 시간이 많이 소요됐다.
-프로그램을 체계적으로 제작하는 방법을 배울 수 있어서 좋았다.
-프로그램을 관리하는 것에 대한 방법을 이해할 수 있었다.

- 후회되는 점도 있고 힘들었지만 배운점이 많아서 좋았습니다

- 소프트웨어공학개론 수업을 들었었는데, 그에 비해 시스템 규모도 크고 문서 작성도 훨씬 복잡해 힘들었지만 남는 게 많은 수업이었던 것 같다. 역시나 첫 단추부터 제대로 끼워야 한다는 것, 모두가 열심히 해야한다는 걸 깨달은 팀플.

- 개발을 할 때, 체계적으로 보고서를 쓰면 좋다는 걸 느꼈다. 특히 다이어그램들. 나중에 꼭 써먹을 것이다.
- 어떻게 개발을 시작해야 하는지 배울 수 있어서 좋았다. 걸음마(requirement 분석)부터 배운 느낌이다.
- 다른 사람과 어떻게 협업해서 코딩을 해야 하는지는 아직도 잘 모르겠다.
- 테스트 케이스를 만들어서 꼼꼼하게 테스트하는데도 의외로 빠뜨린것들이 많았다. 많이 배웠다.

# Q&A

감사합니다