Software Modeling

A NEW C.S.E. ATM

2nd Cycle – System Test

201311299 | 이원오
201311301 | 이재규
201311309 | 전홍준

A NEW C.S.E. ATM

INDEX

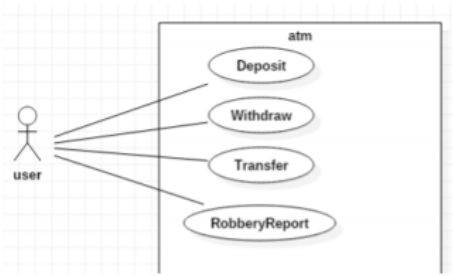
Part 1

Specification Review

Stage 1000

Function	Description
Deposit	User가 사용할 카드나 통장에서 요청한 금액만큼 돈을 받은 뒤, 입금을 해준다
Withdraw	User가 사용할 카드나 통장에서 요청한 금액만큼 돈을 출금해준다.
Transfer	User가 입력한 카드나 통장에서 목적 계좌에 돈을 이체한다.
Language	화면의 인터페이스를 쓰고 싶은 언어로 선택한다.
Check Balance	User가 원하는 카드나 통장의 잔고를 확인한다.
Add Cash	관리자가 요청한 금액만큼 ATM의 현금을 추가한다.
Check Cash	ATM의 현금 잔액을 확인한다.

6.5 Use case diagram



기능 통일

사라진 기능이 남아있고, 있는 기능이 기술되지 않은 문제

불필요한 기능 삭제 및 보고서 수정

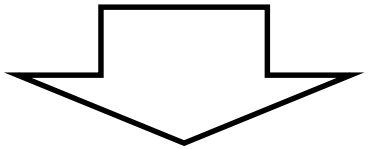
Stage 1004

4. Activity 1004. Record Terms in Glossary

Glossary	Description
Admin	관리자 모드

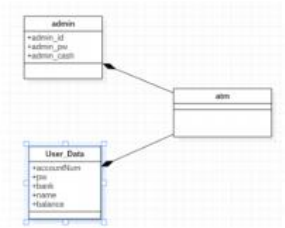
용어 정리

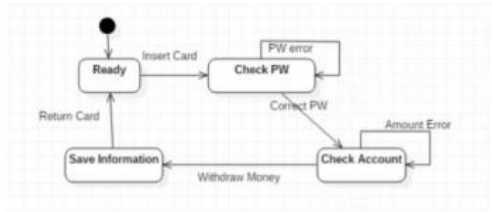
사용하는 용어를 모두 정리하지 않았다.



사용하는 모든 용어 추가

Stage 2030-2037





다이어그램

실제 구현과 맞지 않는 다이어그램

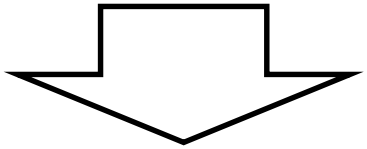
실제로 구현된 프로그램과 일치하게 다이어그램 수정 및 추가.

용어 정리

사용하는 용어에 대한 정확한 설명이 없다.
입력에 대한 정의가 생략되었다.

문서 불완전

생략된 기능, 구현되지 않았는데 삭제되지 않은 기능이 있다.
다이어그램에 대한 수정이 이루어지지 않았다.



문서 불완전 -> 보고서에서 빠져있거나
적절치 않은 부분들을 전체적으로 추가
및 수정 완료.
용어 정리 -> 추가 및 수정 완료.

Part 2

Brute Force Test

1	메뉴 선택 화면에서 잘못된 입력(number)	Success
2	메뉴 선택 화면에서 잘못된 입력(malformed)	Success
3	Database의 계좌 보유액을 조정(1,000,000,000,000,000) 후, 900,000,000,000,000 출금	Failed
4	Database의 계좌 보유액을 조정(1,000,000,000,000,000) 후, 900,000,000,000,000 송금	Failed
5	자기 자신에게 올바른 금액 이체	Success
6	반복 입금으로 ATM 보유액이 long int 범위 이상	Success
7	프로그램 종료, 재시작 후 Database 보존 여부 확인	Success
8	관리자 모드에서 addcash 선택 후, long int 범위 이상의 금액 입금	Failed
9	Deposit 금액 입력 시 long int 범위 이상의 금액 입력	Failed
10	Withdraw 금액 입력 시 long int 범위 이상의 금액 입력	Failed
11	Transfer 금액 입력 시 long int 범위 이상의 금액 입력	Failed

Success 6/11
Failed 5/11

- long int 범위 이상의 int 값이 들어올 경우 에러가 발생한다.
  ->예외처리가 필요하다.
- Specification 명세가 아직 부족하다.
  ->보고서 수정

#	testcases	Result	수정사항	test
1.	메뉴 선택 화면에서 잘못된 입력(number)	Success		Success
2.	메뉴 선택 화면에서 잘못된 입력(malformed)	Success		Success
3.	Database 의 계좌 보유액을 조정(1,000,000,000,000,000) 후, 900,000,000,000,000 출금	Failed	예외처리(보유액의 한도추가)	Success
4.	Database 의 계좌 보유액을 조정(1,000,000,000,000,000) 후, 900,000,000,000,000 송금	Failed	예외처리(송금액의 한도추가)	Success
5.	자기 자신에게 올바른 금액 이체	Success		Success
6.	반복 입금으로 ATM 보유액이 long int 범위 이상	Success		Success
7.	프로그램 종료, 재시작 후 Database 보존 여부 확인	Success		Success
8.	관리자 모드에서 addcash 선택 후, long int 범위 이상의 금액 입금	Failed	예외처리(atm기의 입금가능금액 한도추가)	Success
9.	Deposit 금액 입력 시 long int 범위 이상의 금액 입력	Failed	예외처리	Success
10.	Withdraw 금액 입력 시 long int 범위 이상의 금액 입력	Failed	예외처리	Success
11.	Transfer 금액 입력 시 long int 범위 이상의 금액 입력	Failed	예외처리	Success

Success 6/11
=> 11/11

#	Testcases	Result
101	잘못된 계좌번호(number) 입력	Success
102	잘못된 계좌번호(malformed) 입력	Success
103	올바른 계좌번호 입력했으나, 잘못된 금액(< 0) 입력	Success
104	올바른 계좌번호 입력했으나, 잘못된 금액(malformed) 입력	Success
105	올바른 계좌번호 입력 후, 올바른 금액 입력	Success
201	잘못된 계좌번호(number) 입력	Success
202	잘못된 계좌번호(malformed) 입력	Success
203	올바른 계좌번호 입력 후, 잘못된 비밀번호(number) 입력	Success
204	올바른 계좌번호 입력 후, 잘못된 비밀번호(malformed) 입력	Success
205	올바른 계좌번호와 비밀번호 입력 후, 잘못된 금액(< 0) 입력	Success
206	올바른 계좌번호와 비밀번호 입력 후, 잘못된 금액(> AccountAmount) 입력	Success
207	올바른 계좌번호와 비밀번호 입력 후, 잘못된 금액(> ATMamount) 입력	Success
208	올바른 계좌번호와 비밀번호 입력 후, 올바른 금액 입력	Success
301	잘못된 출금 계좌번호(number) 입력	Success
302	잘못된 출금 계좌번호(malformed) 입력	Success
303	올바른 출금 계좌 입력 후, 잘못된 비밀번호(number) 입력	Success
304	올바른 출금 계좌 입력 후, 잘못된 비밀번호(malformed) 입력	Success
305	올바른 출금 계좌와 비밀번호 입력 후, 잘못된 입금 계좌(number) 입력	Success

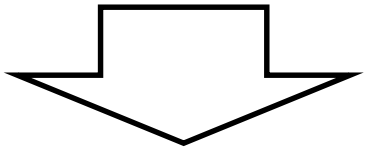
#	Testcase	result
307	올바른 출금 계좌, 비밀번호, 입금 계좌 입력 후, 잘못된 금액(< 0) 입력	Success
308	올바른 출금 계좌, 비밀번호, 입금 계좌 입력 후, 잘못된 금액(< 출금 계좌 보유액) 입력	Success
309	올바른 출금 계좌, 비밀번호, 입금 계좌 입력 후, 잘못된 금액(malformed) 입력	Success
310	올바른 출금 계좌, 비밀번호, 입금 계좌 입력 후, 올바른 금액 입력	Success
401	잘못된 계좌번호 입력	Success
402	올바른 계좌번호 입력 후, 잘못된 비밀번호 입력	Success
403	올바른 계좌번호 입력 후, 올바른 비밀번호 입력	Success
501	잘못된 계좌번호 입력	Success
503	올바른 계좌번호 입력 후, 잘못된 비밀번호 입력	Success
505	올바른 계좌번호 입력 후, 올바른 비밀번호 입력	Success
601	잘못된 관리자 아이디 입력	Success
602	올바른 관리자 아이디를 입력 했으나, 잘못된 관리자 비밀번호(number) 입력	Success
603	올바른 관리자 아이디를 입력 했으나, 잘못된 관리자 비밀번호(malformed) 입력	Success
604	올바른 관리자 아이디 비밀번호를 입력 했으나, 잘못된 금액(< 0) 입력	Success
605	올바른 관리자 아이디 비밀번호를 입력 했으나, 잘못된 금액(malformed) 입력	Success
606	올바른 관리자 아이디, 비밀번호, 금액 입력	Success
701	잘못된 관리자 아이디 입력	Success
702	올바른 관리자 아이디를 입력 했으나, 잘못된 관리자 비밀번호(number) 입력	Success
703	올바른 관리자 아이디를 입력 했으나, 잘못된 관리자 비밀번호(malformed) 입력	Success
704	올바른 관리자 아이디, 비밀번호 입력	Success

Part 3

Static Analysis

JAVA_66 - 반복문 안에서 string concatenation assignment 연산자의 사용 금지

```
452         for(i =0; i< len; i++)
453         {
454
455             if((l[i].substring(0, 6)).equals(account))
456             {
457                 l[i] = d.getAccountNum() +" " + d.getPw()+" " + d.getBank()+" " + d.getName()+" " + d.getBalance();
458
459             }
460         }
```



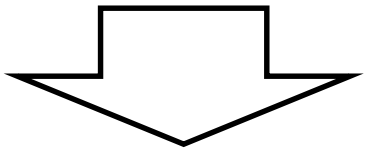
```
for(i =0; i< len; i++)
{

    if((l[i].substring(0, 6)).equals(account))
    {
        l[i] = "";
        l[i] = l[i].concat(d.getAccountNum());
        l[i] = l[i].concat(" ");
        l[i] = l[i].concat(d.getPw());
        l[i] = l[i].concat(" ");
        l[i] = l[i].concat(d.getBank());
        l[i] = l[i].concat(" ");
        l[i] = l[i].concat(d.getName());
        l[i] = l[i].concat(" ");
        l[i] = l[i].concat(d.getBalance());
    }
}
```

String 메소드의 기능 중 한 개인 concat으로 변경

JAVA_64 - 반복문 안에서 string concatenation assignment 연산자의 사용 금지

```
417         for(i = 0; i < len; i++)
418         {
419             if(i == len-1)
420             {
421                 line += l[i];
422             }
423             else
424             {
425                 line += l[i];
426                 line += "\r\n";
427             }
428         }
```

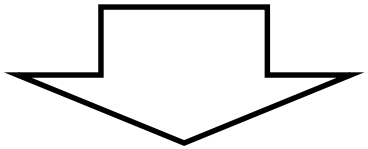


```
if(i == len-1)
{
    line = line.concat(l[i]);
}
else
{
    line = line.concat(l[i]);
    line = line.concat("\r\n");
}
```

String 메소드의 기능 중 한 개인 concat으로 변경

JAVA_66 - 조건문에 선언 연산자 사용 금지

```
481    public static boolean calculate(User_Data d, int money)
482    {
483        int balance = Integer.parseInt(d.getBalance());
484
485        if((balance += money) < 0)
486        {
487            return false;
488        }
489        else
490        {
491            d.setBalance(Integer.toString(balance));
492            return true;
493        }
494
495    }
```



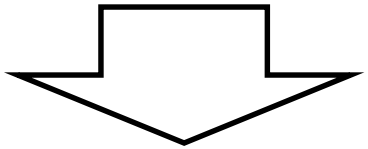
```
int balance = Integer.parseInt(d.getBalance());
balance += money;
if(balance < 0)
{
    return false;
}
else
{
    d.setBalance(Integer.toString(balance));
    return true;
}
```

연산식을 조건문 밖에서 수행

Sun_26 - 괄호 안의 수식에 연산자 혼용 금지

```
if(amount.charAt(i) >= 48 && amount.charAt(i) <= 57)
```

```
if(amount <= 1000000 && amount > 0)
```



```
if((amount.charAt(i) >= 48) && (amount.charAt(i) <= 57))
```

연산자 혼용 방지를 위한 괄호 추가

```
if((amount <= 1000000) && (amount > 0))
    return true;
else
    return false;
```


```java
public static void transfer(Admin admin)
{
    String account = getAccount();
    User_Data from = null;
    try {
        from = find(account);
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
```

Throw IOException 대신 try catch 구문 추가.

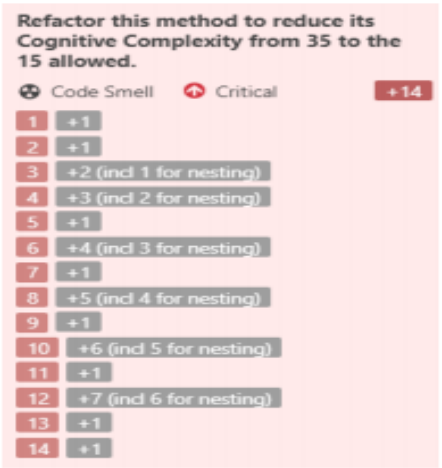
01

02

03

04

중첩된 if 문을 줄일 것



너무 복잡한 함수

```
public static void transfer(Admin admin)
{
    String account = getAccount();
    User_Data from = null;
    try {
        from = find(account);
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    if(from == null)
    {
        System.out.println("cannot find the account!");
        return;
    }
    String pw = getPw();
    if(from.checkAccountPw(account, pw))
    {
        String to_account = getTransferAccount();
        if(to_account.equals(account))
        {
            System.out.println("자기 자신의 계좌로는 이체할 수 없습니다.");
            return;
        }
        User_Data to = null;
        try {
            to = find(to_account);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        if(to == null)
        {
            System.out.println("Wrong Account!");
            return;
        }
            int amount = getAmount();
            if(amount < 0)
            {
                System.out.println("잘못된 금액 입력");
                return;
            }
                if(Integer.parseInt(from.getBalance()) < amount)
                {
                    System.out.println("not enough money");
                    return;
                }
```

Transfer 메소드에 다수로 중첩되어 있는 if문을 적절히 return을 통해서 복잡성 해소.

Part 4

Conclusion

예외 처리

구현 완성

Specification 재작성

느낀점

Software Modeling

A NEW C.S.E. ATM

Thank you

201311299 | 이원오
201311301 | 이재규
201311309 | 전홍준