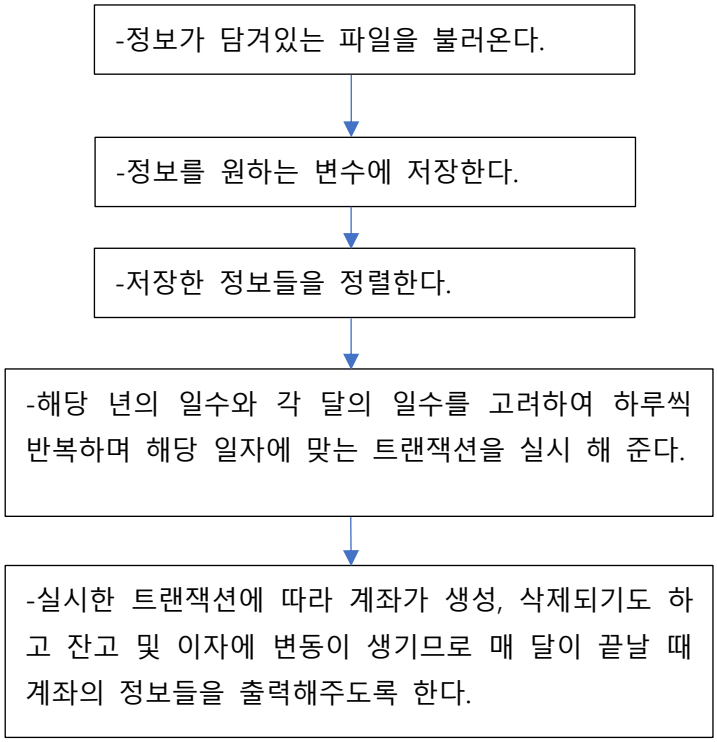
객체지향 프로그래밍
2차 프로그래밍 과제
(OOP-HW2-201721083)

이름 : 김태석
학번 : 201721083
학과 : 미디어학과
학년 : 2학년

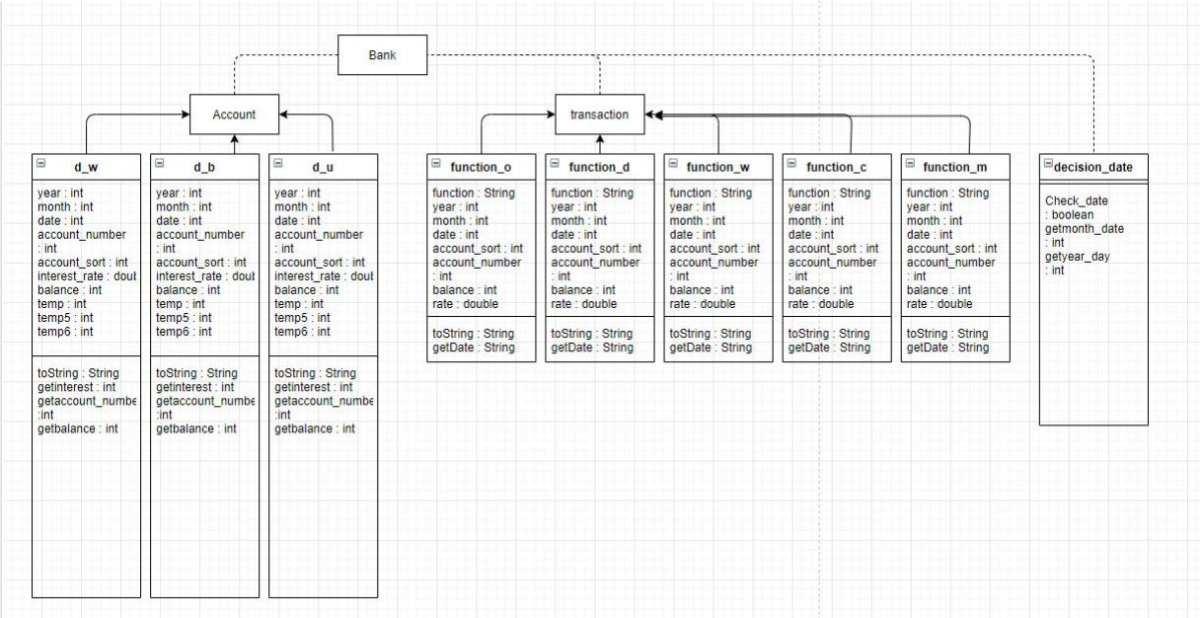
나)소개

구현 목록	구현 여부(O/X)
Inheritance 사용	O
Polymorphism 사용	O
입력 데이터 유효성 검사	O
트랜잭션 정렬시 Arrays.sort 메소드 사용	O
Lamda Expression 사용	O
계좌 개설 전 입금 오류메세지	O
계좌 개설 전 출금 오류메세지	O
계좌 개설 전 해지 오류메세지	O
계좌 잔액 부족시 출금 금지 오류메세지	O
트랜잭션의 지정된 명령어 이외의 명령어가 있을 경우 오류메세지	O
이율이 0~100 사이의 값이 아닐 경우 오류메세지	O
1~3 이외의 숫자가 계좌의 종류로 포함 되어있을 경우 오류메세지	O
계좌 번호가 1000~9999 사이의 값이 아닐 경우 오류메세지	O

기본 로직



다) Class Diagram



Class

-Account

d_u,d_b,d_w : 계좌가 개설되는 날짜를 년 월 일 에 따라서 인트값으로 받는다. 각 계좌에 해당하는 계좌번호를 인트값으로 받고, 각 계좌의 종류에따라 account_sort값을 다르게하고, 계좌 개설 시 입금액이 balance가 된다. 이 밸런스는 수시입출금 계좌에서 입출금에 따라 금액이 변경 될 수 있다.

temp가 변수로 들어가 있는 이유는 계좌가 개설 될 때, 혹은 이자율이 변경 될 때 한달 중 변경되기 이전까지의 이자들을 잠시 저장한 후 나중 이자 계산 때 빼주기 위함이다.

-transaction

Function_o : 계좌를 개설하는 기능을 수행한다. 저장되어 있는 값에 따라서 초기 잔고, 이자율, 계좌 번호 및 개설 날짜, 계좌 종류를 저장하는 값을 지닌다.

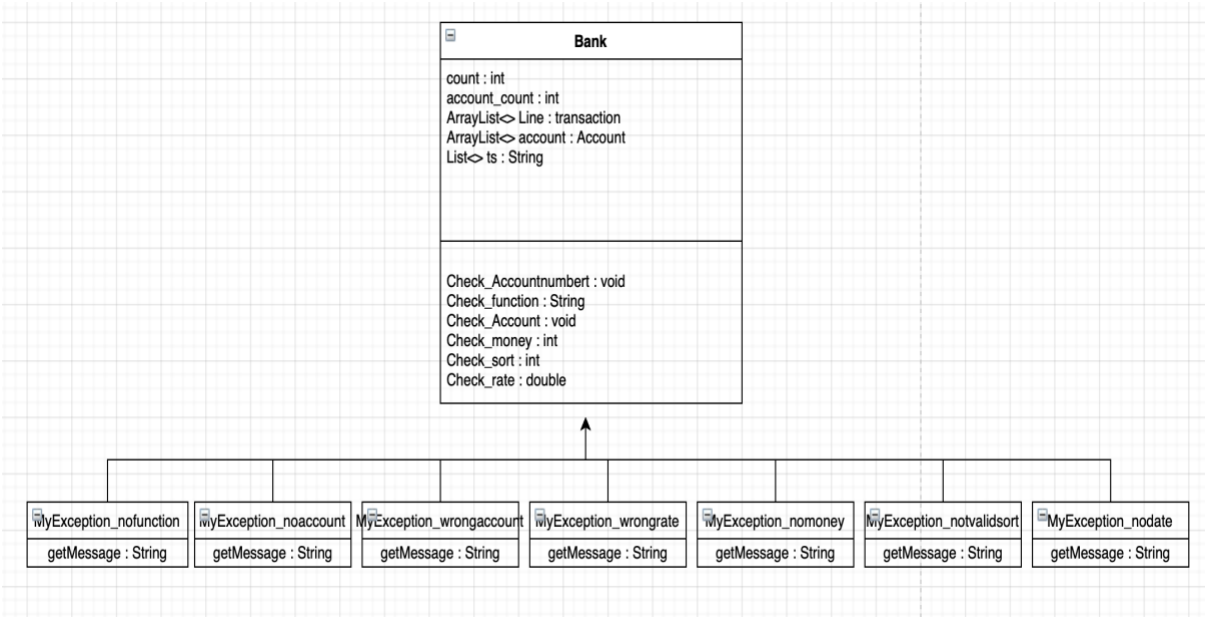
Function_d : 예금을 하는 기능을 수행한다. 예금하는 날짜를 저장하고, 예금하는 계좌번호를 받아온다. 예금액을 deposit_money를 통해 저장하고 해당 계좌에 더해준다.

Function_w: 출금을 하는 기능을 수행한다. 출금하는 날짜를 저장하고, 출금하는 계좌번호를 받아온다. 출금액을 withdraw_money를 통해 저장하고 해당 계좌에서 빼준다.

Function_c: 계좌를 해지하는 기능을 수행한다. 해지하는 날짜를 저장하고, 해지하는 계좌번호를 받아온다.

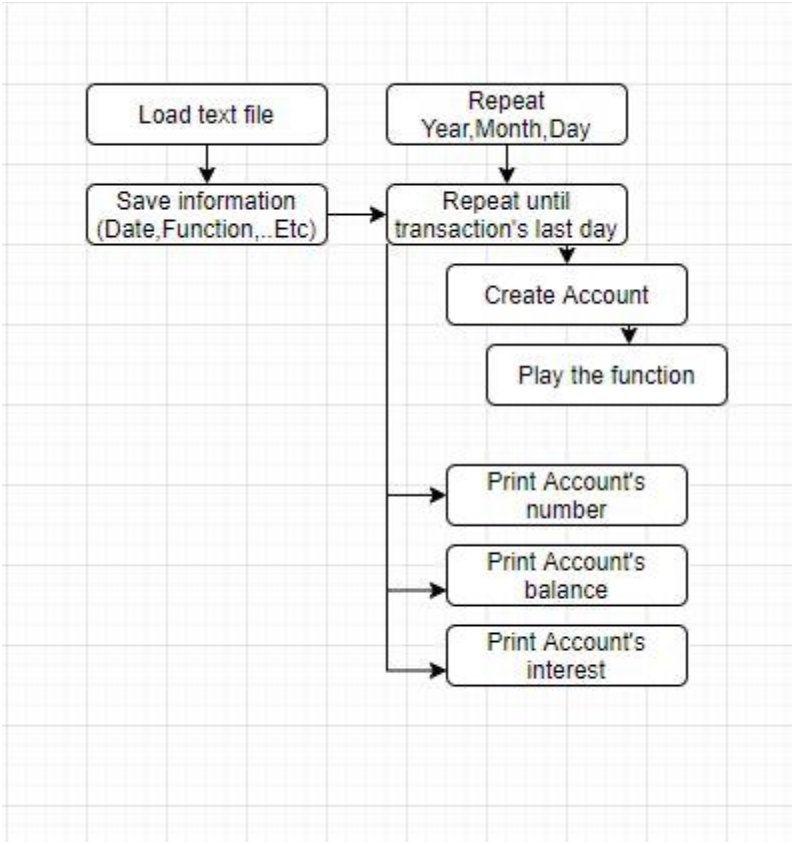
Function_m: 이자율을 변경하는 기능을 수행한다. 이자율 변경 날짜를 저장하고, 이자율 변경을 하는 계좌번호를 받아온다. 이자율을 rate에 저장하여 해당 계좌 이자율을 변경하게 된다.

-decision_date:
해당년도가 윤년인지 아닌지를 판별하여 각 달의 날짜를 계산해주는 getmonth_date 메소드가 있고, 각 년도의 일수를 리턴해주는 getyear_day 메소드가 있다.
또한 입력받은 날짜가 실제 달력에 존재하는지 아닌지를 판별하는 Check_date 메소드가 존재한다.



-Bank :
List<String> ts : 트랜잭션 파일을 일시적으로 저장해준다. 이는 나중에 ArrayList<transaction> Line에 각 기능에 알맞게 저장된다.
ArrayList<transaction> Line : 트랜잭션 각 한줄에 대한 정보를 기능에 알맞게 저장해준다. 기능에 맞는 transaction 의 자식클래스들을 객체로 생성하며 추가해 나간다.
ArrayList<Account> account : 트랜잭션을 실시하면서 계좌 개설에 대한 기능이 실시될 때 마다 트랜잭션에 대한 값에 맞게 계좌를 개설한다. Account 의 자식클래들은 d_u,d_b,d_w 3가지 객체를 알맞게 생성하며 추가해 나간다.
Method
Check_Accountnumber : 계좌번호가 1000~9999가 아니면 트라이 캐치를 이용하여 오류메세지가 출력되도록 한다.
Check_function : 기능이 d,w,o,c,m 외에 것이 들어가있다면 트라이 캐치를 이용하여 오류메세지가 출력되도록 한다.
Check_Account : 계좌가 개설되지 않았다면 트라이 캐치를 이용하여 오류메세지가 출력되도록한다.
Check_money : 계좌에 출금을 진행 할 때 잔고-출금액 이 0보다 작다면 트라이캐치를 이용하여 오류메세지가 출력되도록 한다.
Check_sort : 계좌의 종류가 1~3 이외의 값이 입력되어있다면 트라이캐치를 이용하여 오류메세지가 출력되도록 한다.
Check_rate : 계좌의 이자율이 0~100 사이의 실수값이 아니라면 트라이캐치를 이용하여 오류메세지가 출력되도록한다.

다)Activity Diagram



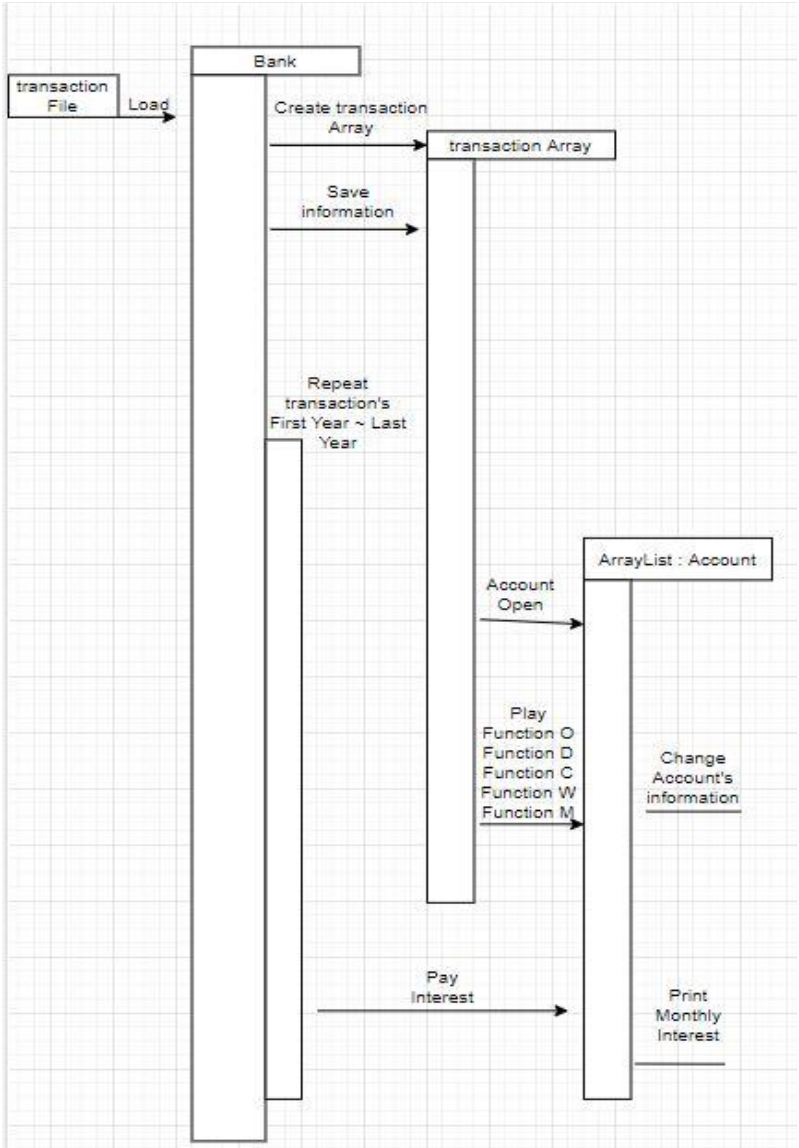
-설명-

텍스트 파일에서 정보들을 불러들여와서 값을 알맞는 위치에 저장한다.

그 후 저장된 정보를 정렬하고 정렬된 값 중 가장 빠른 년 월 일 값부터 트랜잭션을 실행 시킬 수 있도록 한다. 트랜잭션은 정렬된 값 중 가장 늦은 년 월 까지 반복한다.

그 사이 매 달 알맞는 계좌에 이자를 지급 하고, 반복문이 돌아가며 해당하는 날짜와 일치하는 트랜잭션 날짜가 있으면 트랜잭션의 명령어에 따라서 기능을 실행시킨다.

다)Sequence Diagram



-트랜잭션이 무작위로 저장되어 있는 파일을 불러들여온다.

-트랜잭션 어레이를 만들어서 각 기능(d,o,w,c,m)에 알맞게 값들을 저장 해 준다.

-저장된 값 중 날짜를 이용하여 1차적으로 정렬을 해 주고 날짜가 같다면 계좌번호 순서대로 정렬을 실시한다.

-정렬 된 값들을 이용하여 반복문을 실시한다. 반복문은 정렬된 트랜잭션의 첫줄에 해당하는 날짜부터 마지막 줄에 해당하는 월까지 반복된다.

-Account를 담는 ArrayList를 만들어 주어 트랜잭션이 실시되면서 계좌를 개설 할 때마다 ArrayList에 추가해준다.

-트랜잭션에 해당하는 날짜와 반복문 속의 날짜가 일치하면 트랜잭션을 실시 해 주고 그에 따라 계좌의 정보가 변경되면 계좌 정보를 변경 해 준다.

-매 달이 끝나 다음달로 넘어가기 전 각 계좌의 잔액, 이자, 총 금액을 출력 해 준다.

라) 실행 화면

```
Hello. Welcome to AJOU BANK!

ENTER YOUR TRANSACTION FILE NAME
trans.txt
 Before Sorting :
d 2020 5 10 1234 100000
o 2020 3 1 1 1234 2000000 0.2
o 2020 1 31 2 2222 10000000 2.5
w 2020 5 15 1234 200000
o 2020 2 1 3 3333 10000000 2.0
w 2020 4 15 1234 200000
m 2020 4 15 3333 2.2
c 2020 4 20 2222
d 2020 2 20 1234 100000

 After Sorting :
o 2020 1 31 2 2222 10000000 2.5
o 2020 2 1 3 3333 10000000 2.0
d 2020 2 20 1234 100000
o 2020 3 1 1 1234 2000000 0.2
w 2020 4 15 1234 200000
m 2020 4 15 3333 2.2
c 2020 4 20 2222
d 2020 5 10 1234 100000
w 2020 5 15 1234 200000
[2020]
MONTH : [1]
#2222 10000000      683 10000683
MONTH : [2]
Account #1234 is not exist
#2222 10000000    20491 10020491
#3333 10000000    15846 10015846
MONTH : [3]
#1234  2000000      338  2000338
#2222 10000000    41665 10041665
#3333 10000000    32812 10032812
MONTH : [4]
#2222 10000000    55326 10055326
#1234  1800000      648  1800648
#3333 10000000    50136 10050136
MONTH : [5]
#1234  1700000      944  1700944
#3333 10000000    68863 10068863
```

마)결론

프로그래밍을 하며 배운 점

1. 파일 입출력 및 파일 속 문자 복사
   파일 속에 저장되어있는 문자들을 복사해서 활용하는 방법을 배웠다. 스플릿을 이용하여 띄어쓰기 단위로 각 값들을 저장하고 트랜잭션의 한 줄에서 맨 앞부분에 위치한 명령어들에 따라서 필요한 값을 원하는 객체에 저장하였다.

2. ArrayList의 활용
   ArrayList를 사용하여 기존에 쓰던 배열보다 훨씬 수월하게 작업을 하였다.
   기존의 배열은 크기를 미리 선언을 해 주어야지만 예외가 발생하지 않고 코드가 돌아갔지만, ArrayList를 사용하여 배열의 크기를 미리 선언 해 주지 않아도 필요할 때 add메소드를 이용하여 확장시켜나가고 필요하다면 remove메소드를 사용하여 ArrayList를 손쉽게 수정 할 수 있었다.

3. 정렬
   과제를 수행하며 정말 다양한 방식의 정렬법이 있다는 것을 배웠다.
   다중 정렬을 이용하여 내가 원하는 순서대로 배열들을 정렬해 주는 방법을 배웠고, Comparable 을 implements하여서 원하는 정렬을 설정 해 주는 법을 익히게 되었다.

4. 예외처리
   지정된 값 이외의 값이 저장 되어 있거나, 혹은 올바르지 못한 값들이 파일 속에 저장되어 있을 수도 있다. 가령 날짜가 올바르지 못하거나, 계좌에 잔액이 부족한데 출금을 할 수 있다면 아마 실제 프로그램에선 크나큰 오류일 것이다. 이전 1차 프로그래밍 과제에선 예외처리를 try catch와 throw 등의 기능을 사용하지 않고 예외인 상황을 모두 조건문으로 만들어 처리 해 주었다면, 이번 2차 프로그래밍 과제에선 각종 예외들을 미리 설정해 주고, try catch를 이용하여 예외가 발생하였다면 해당 명령어에 대한 내용을 실행해 주지 않고 예외가 발생하였음을 메세지로 알릴 수 있었다.

5. 상속 및 다형성 활용
   부모 클래스와 자식클래스들을 작성하여, 이를 ArrayList에서 사용하였다.
   ArrayList를 처음 선언해줄 때 부모클래스로 선언을 해주었고, 값들을 add할 때는 자식클래스의 객체들을 생성하여 넣어주었다. 이를 통해서 한 ArrayList에서 다양한 종류의 계좌를 한꺼번에 관리할 수 있었다.

프로그래밍 시 어려웠던 점

1. 값들이 입력되어 있는 파일 내용 저장
   파일의 내용을 저장할 때 모두 문자열로 저장이 되어서 각 자료형에 맞게끔 모두를 변환해서 저장 해 주어야 했다. 또한 파일 속 내용이 모두 일정길이, 형식을 갖춘게 아니라 지정된 명령어에 따라서 인자로 받아오는 수가 다 달랐기 때문에 각 줄 맨처음 저장되어 있던 명령어에 알맞게 값들을 저장해 주어야했다.

2. 날짜순 정렬 후 계좌번호 순 정렬.
   다양한 정렬방법이 있지만, 이중으로 정렬하는 방법은 생각 해 보지를 않았다. 아직 람다 표현식이 익숙하지않아서 하루 정도를 트랜잭션 정렬하는 데에 할애 하였다. 생각해보면 굉장히 단순한 논리인데 미숙하여서 시간이 오래 걸렸던 것이 많이 아쉬웠다.

3. 이자 계산과 지급
   트랜잭션을 한줄씩 읽을때마다 그에 따른 명령을 실행시키는 것은 별 큰 문제가 되지 않았지만, 명령을 수행하고 매달 이자를 지급하는 것을 구현해 내기가 어려웠다.
   다양한 명령어에 따라서 은행 잔고가 변경 될 수 있고, 이율까지 변경 되면 변경된 잔고에 따라서 지급해야 하는 이자가 달라져서 이를 해결하는 것이 문제였다. 때문에 가장 시간이 오래 걸린 부분이기도 하다. 이자가 예시와 계속 다르게 나와서 코드를 수차례 읽으며 잘못 된 부분을 수정해야 했다. 결국 이자를 매달 지급하기 위해서 1년중 12개월이 반복되게끔만 설정해 주었던 부분을 각 달의 일수를 구하여 하루하루 반복해주는 설정으로 바꾸었고, 그에 따라 여러 조건들도 추가 해 주어야 했다.

4. 예외처리
   문제에서 제시된 예외처리 말고도 생각해야 하는 예외가 더 있다고 생각했다.
   명령어가 올바르지 않은 경우, 계좌번호가 이상한 경우, 이율이 이상한 경우 등등.. 이런 예외들 하나하나를 다 잡아내야만 완벽한 프로그램이 완성 될 수 있다는 생각에 여러 상황을 계속해서 고민하고 그에 따른 예외를 추가해 주었다.
   또한 try catch문이 예외를 잡아내서 예외처리를 해준다고 하여도, 그 후의 일들은 계속해서 진행이 되어야 하기 때문에 언제 try catch로 예외를 잡아줄 지 일일이 생각해 내야 했다. 하지만 코드를 작성하면서 예외가 발생할 수 있는 부분과 아닌 부분을 나누니 조금은 쉽게 해결 할 수 있었다. 공부를 하며 알게 된 점인데 try catch를 많이 사용하면 코드가 길어질 뿐더러 메모리가 많이 사용되어 안좋다고 한다. 그렇기에 최대한 필요한 부분에만 사용하려 노력하였다.