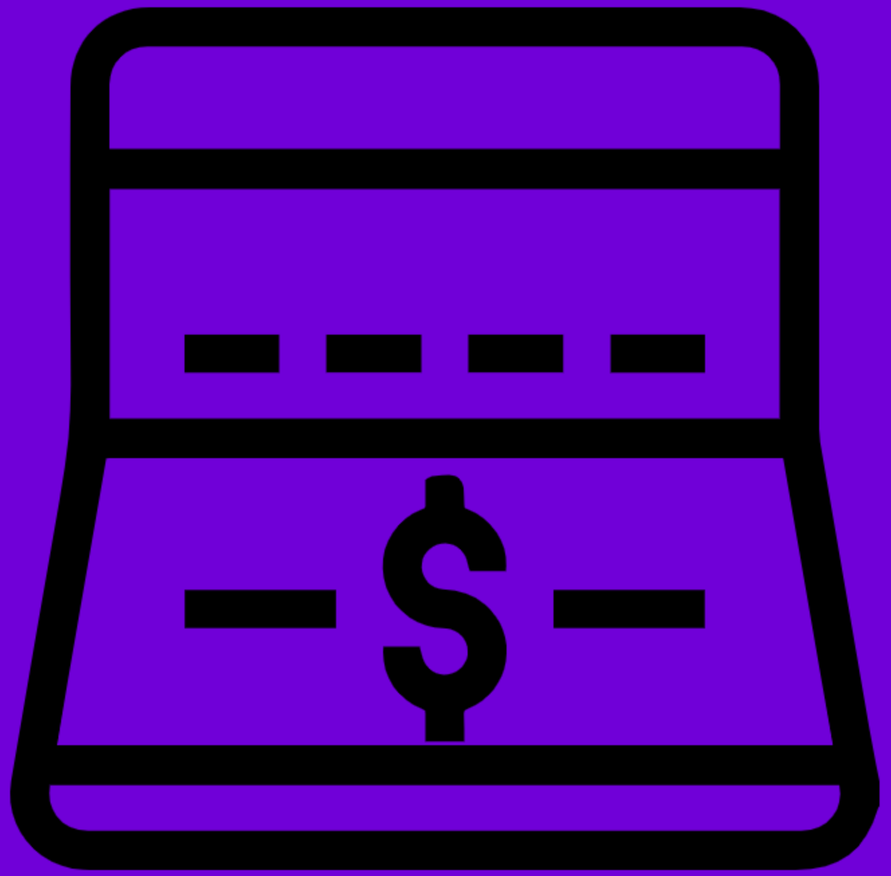
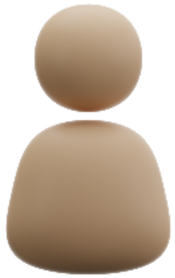
계좌 관리 프로그램

교육과정 1개월차 (12월28일~1월19일)



신연재

Contents.



프로젝트 개요

본 프로그램은 JAVA강의 4주 수료후 만든
간단한 계좌관리 프로그램 입니다.
프로그램의 사용자는 등록된 계좌를 조회하고 삭제하고
생성을 할 수 있습니다. 또한 원하는 금액을 입금하고 마이너스 계좌를 이용
대출을 할 수 있습니다.

주요 기술

코드구현 - JAVA
화면구현- AMS

1. 자바의 Static과 this의 개념
2 service클래스를 만들어서 구현
3. 객체 추상화 및 생성자 오버로딩 활용
4. Validator 클래스를 만들어서 유효성 검증

1. 화면 구현 방법으로 AWT의 FRAME사용

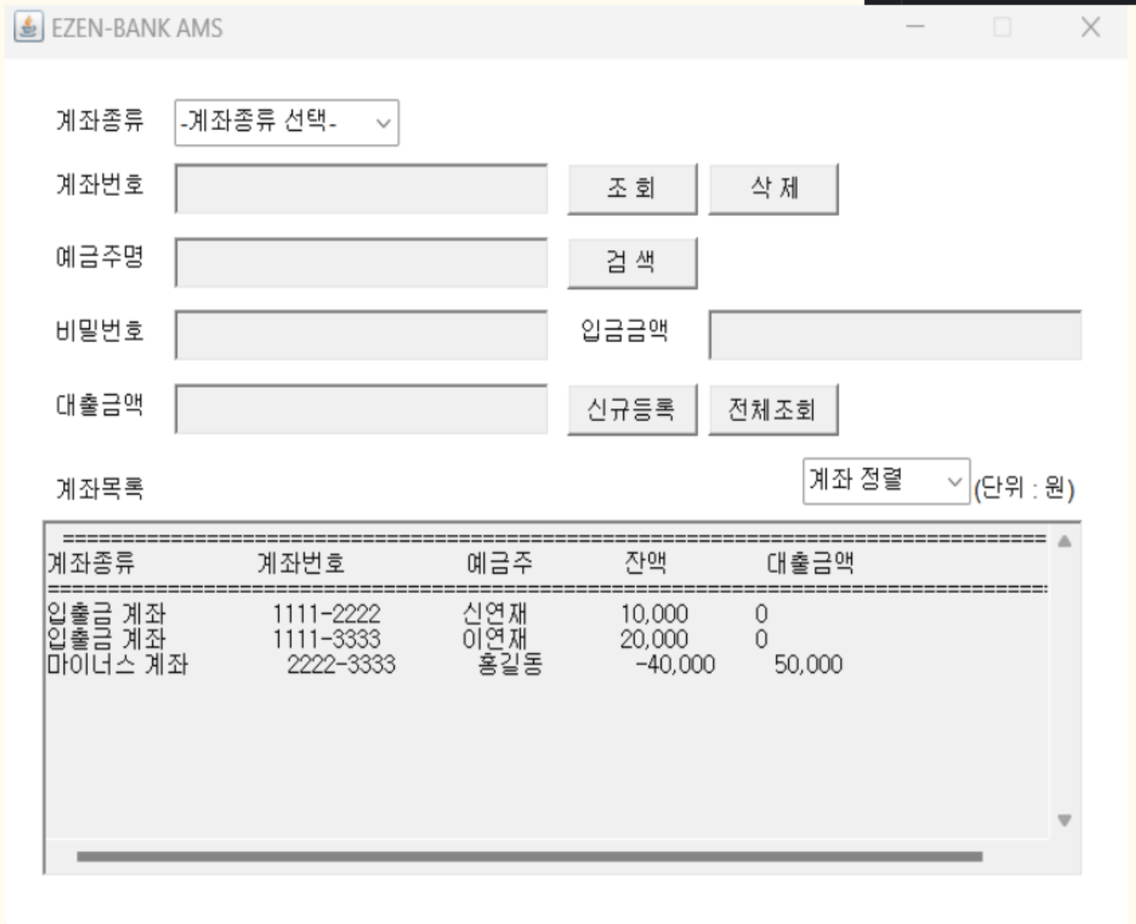
요구분석

1. 사용자는 계좌 종류로 입출금계좌와 마이너스 계좌를 선택 할 수 있다.
2. 사용자는 계좌번호를 입력하여 계좌를 조회 할 수 있다.
3. 사용자는 계좌번호와 예금주명 비밀번호 입금금액을 입력하여 새로운 계좌를 생성 할 수 있다.
4. 사용자는 계좌번호를 입력하여 계좌를 삭제 할 수 있다.
5. 사용자는 전체조회 버튼을 이용하여 모든 계좌를 볼 수 있다.
6. 사용자가 마이너스 계좌를 선택 했을 경우 화면 잔액표시란에 마이너스로 기록이 된다.
7. 프로그램 종료 시 종료를 물어보는 창이 나오고 취소를 누르면 뒤로가고 종료를 누르면 프로그램이 종료가 된다.

실행 화면

```
public BankFrame(String title){  1 usage
    super(title);
    accountService = new AccountService();
    accountService.addAccount(new Account( accountNum: "1111-2222", accountOwner: "신연재", passwd: 1111, balance: 10000));
    accountService.addAccount(new Account( accountNum: "1111-3333", accountOwner: "이연재", passwd: 1111, balance: 20000));
    accountService.addAccount(new MinusAccount( accountNum: "2222-3333", accountOwner: "홍길동", passwd: 2222, balance: 10000, borrowMoney: 50000));
```

초기 실행 화면



1. 처음 실행이 되면 기존에 넣어두었던 더미데이터 출력.

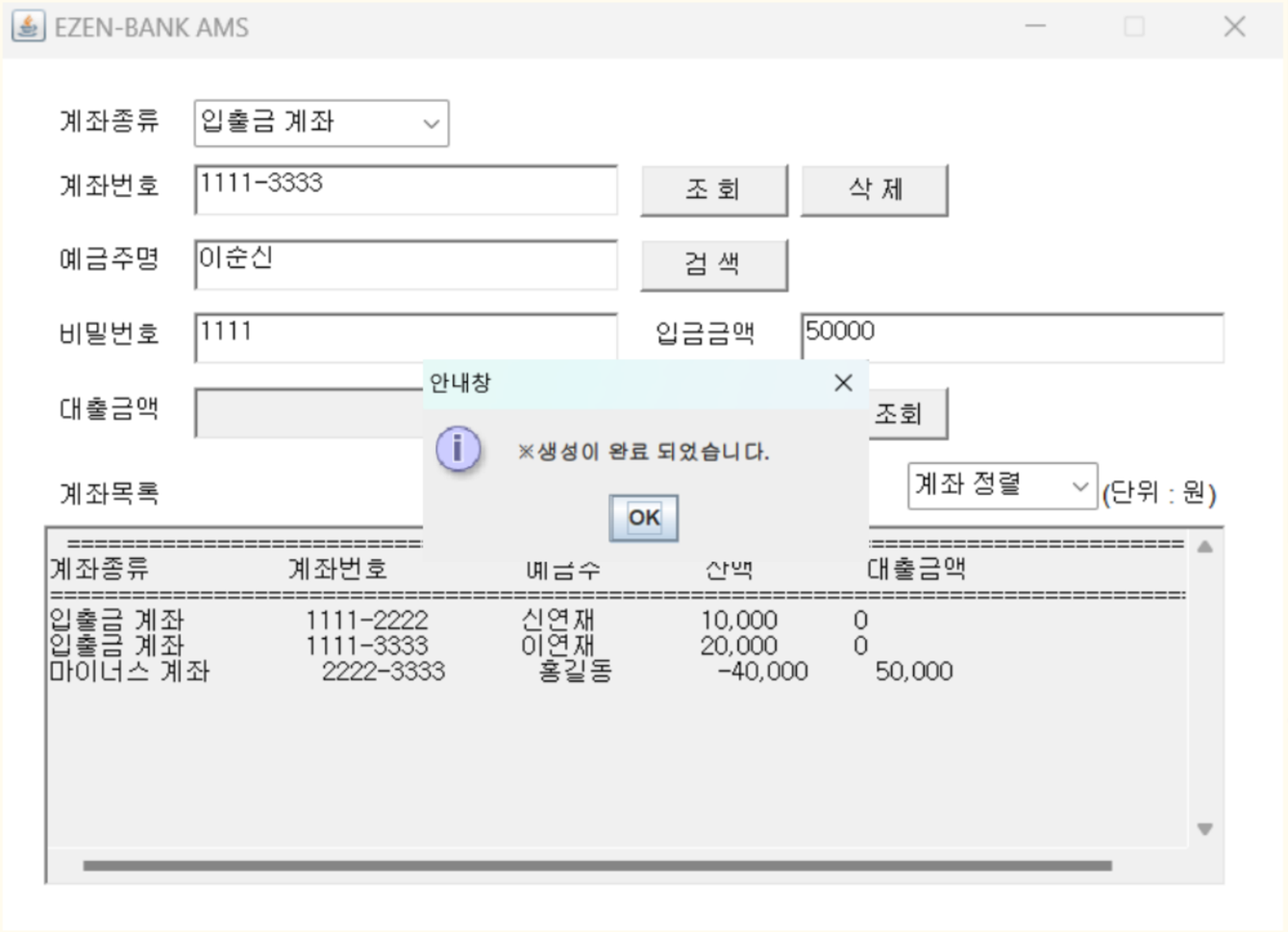
AWT로 화면 구성

```
accountNumLabel = new Label( text: "계좌번호");
accountNumTF = new TextField();
accountOwnerLabel = new Label( text: "예금주명");
accountOwnerTF = new TextField();
passwdLabel = new Label( text: "비밀번호");
passwdTF = new TextField();
depositMoneyLabel = new Label( text: "입금금액");
depositMoneyTF = new TextField();
borrowMoneyLabel = new Label( text: "대출금액");
borrowMoneyTF = new TextField();
checkButton = new Button( label: "조 회");
deleteButton = new Button( label: " 삭 제");
searchButton = new Button( label: "검 색");
newAccountButton = new Button( label: "신규등록");
checkAllButton = new Button( label: "전체조회");
accountListLabel = new Label( text: "계좌목록");
listTA = new TextArea();
accountListSort = new Choice();
accountListSort.add("계좌 정렬");
unitLabel = new Label( text: "(단위 : 원)");
listTA.setBackground(new Color( r: 240,  g: 240,  b: 240));
```

실행 화면

새로운 계좌 생성 화면

1. 사용자가 각 InputBox에 알맞은 데이터 입력
2. 신규등록 버튼을 누르면 생성이 완료되었다는 안내창 출력




```java
/**
 * 계좌번호, 예금주, 비밀번호, 잔액 필드 생성
 */

private String accountNum;  4 usages
private String accountOwner;  4 usages
private int passwd;  5 usages
private long balance;  8 usages
```

```java
public void addAccount(Account account) {
    this.accounts.put(account.getAccountNum(), account);
}
```

1. 사용자가 입력하는 값인 계좌번호, 예금주 , 비밀번호, 잔액 필드 생성
2. AccountService클래스에 새로운 계좌를 객체로 등록

실행 화면

```
accountNumber = accountNumTF.getText();
accountOwner = accountOwnerTF.getText();
String inputPasswd = passwdTF.getText();
String inputMoney = depositMoneyTF.getText();

if (!Validator.isText(accountNumber)){
    fieldError(accountNumTF, errorMessage: "계좌번호는 필수입력 사항입니다.");
}
if (!Validator.isText(accountOwner)){
    fieldError(accountOwnerTF, errorMessage: "예금주명은 필수입력 사항입니다.");
}
if(!Validator.isText(inputPasswd)){
    fieldError(passwdTF, errorMessage: "필수입력 항목입니다.");
}

if(!Validator.isText(inputMoney)){
    fieldError(depositMoneyTF, errorMessage: "필수입력 항목입니다.");
}

/**
 * 입력된 정보의 2차 검증 : 유효한 입력 형식인지 검증
 */
if(!Validator.isNumber(inputPasswd)){
    fieldError(passwdTF, errorMessage: "숫자형식이어야 합니다.");
}

if(!Validator.isNumber(inputMoney)){
    fieldError(depositMoneyTF, errorMessage: "숫자형식이어야 합니다.");
}
password = Integer.parseInt(inputPasswd);
restMoney = Long.parseLong(inputMoney);

JOptionPane.showMessageDialog( parentComponent: null, message: "※생성이 완료 되었습니다.", title: "안내창",JOptionPane.INFORMATION_MESSAGE);
```

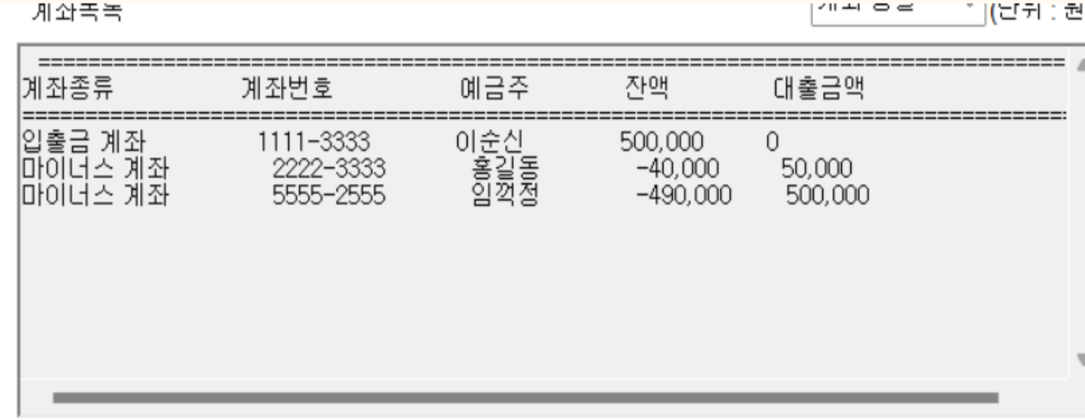
```
public static boolean isText(String value) { return value != null && value.trim().length() != 0; }

public static boolean isId(String inputId) {
    if (inputId == null) {
        return false;
    } else {
        for(int i = 0; i < inputId.length(); ++i) {
            char ch = inputId.charAt(i);
            if (!Character.isAlphabetic(ch) && !Character.isDigit(ch)) {
                return false;
            }
        }

        return true;
    }
}
```

1. 유효성 검증을 위해 Validator 클래스를 생성
2. Validator 클래스를 활용하여 사용자 유효성 검증 완료

실행 화면

마이너스 계좌 생성 화면

1. 사용자가 InputBox에 대출금액 포함한 알맞은 데이터 입력
2. 신규등록 버튼을 누르면 마이너스 계좌 생성 완료

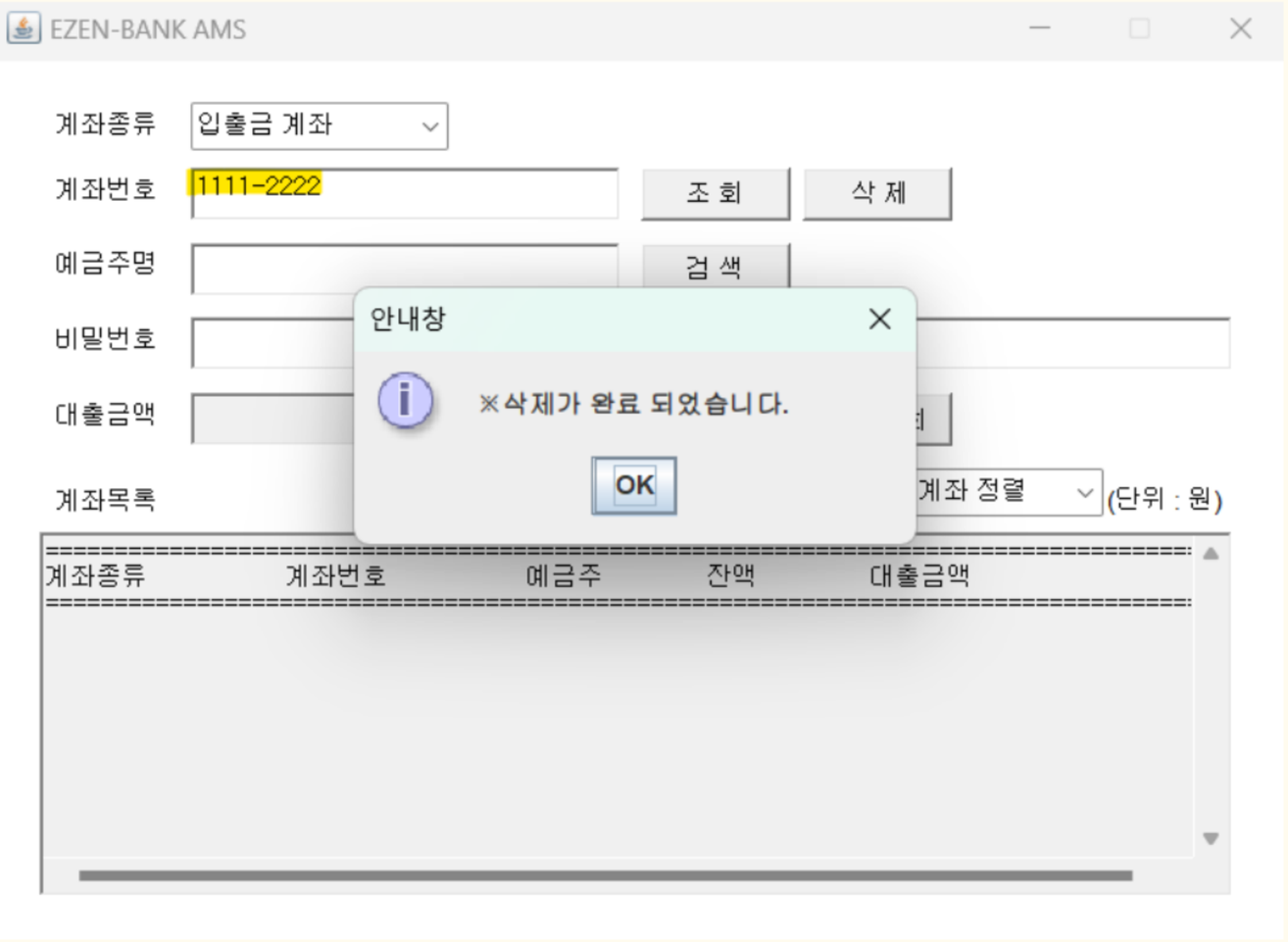


마이너스 계좌가 등록 되고 잔액은 대출금액 - 잔액으로 계산된다.

실행 화면

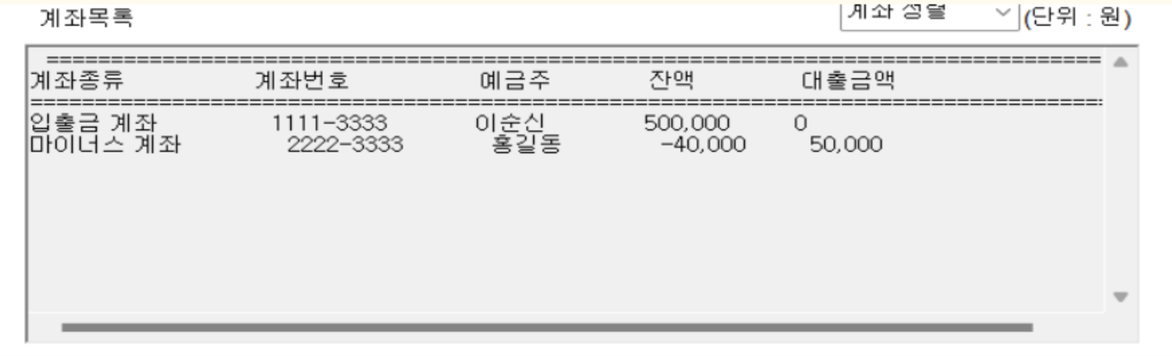
계좌 삭제 화면

1. 사용자가 알맞은 계좌번호 데이터 입력
2. 삭제 버튼을 누르면 삭제가 완료되었다는 안내창 출력
3. 계좌 목록에서 계좌번호 삭제




```java
public boolean removeAccount(String accountNum) {
    Account account = (Account)this.accounts.remove(accountNum);
    return account != null;
}
```

AccountService클래스에서 전체 배열로 받은 account를 remove매서드를 이용하여 삭제

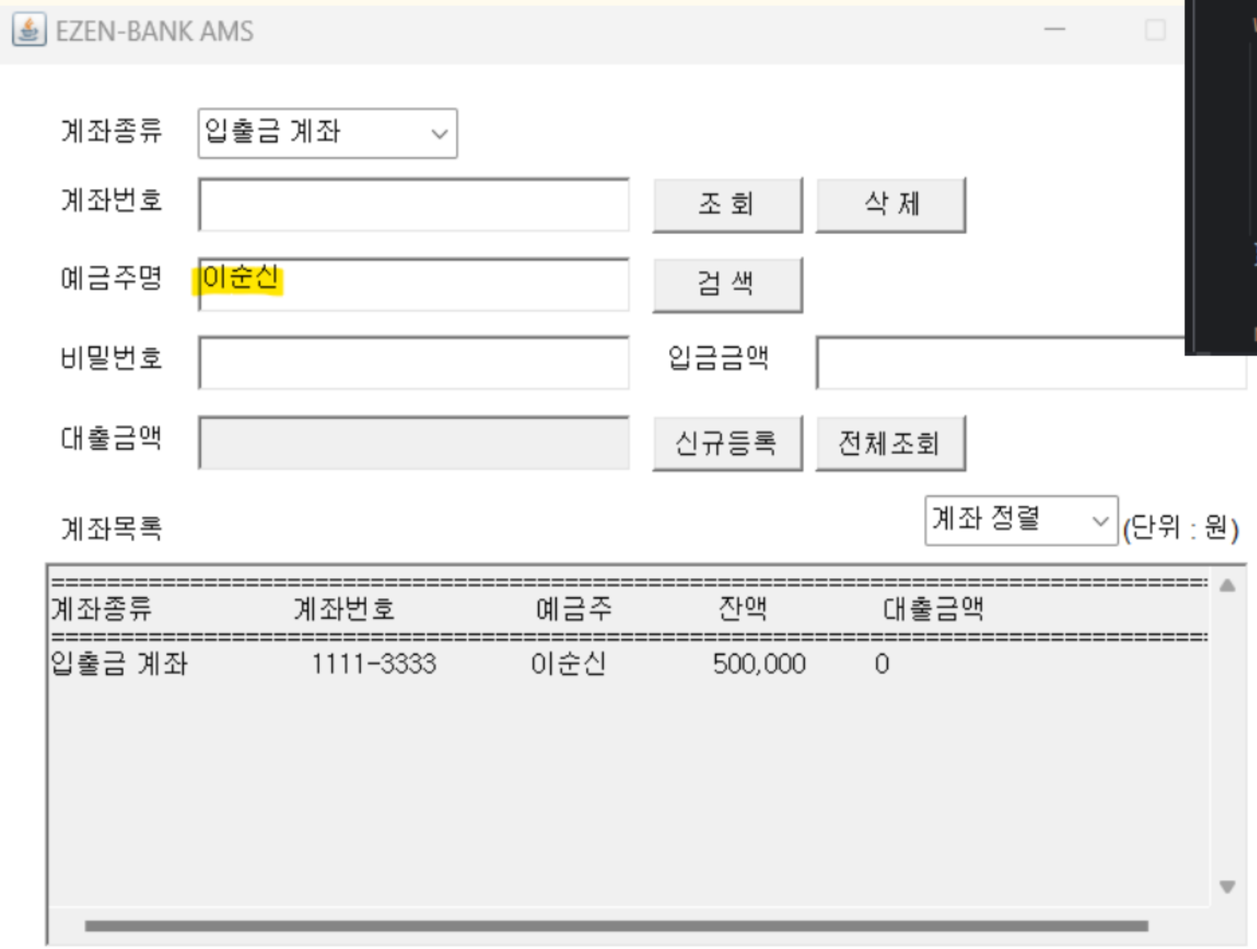


사용자가 삭제한 계좌가 계좌목록에서 삭제

실행 화면

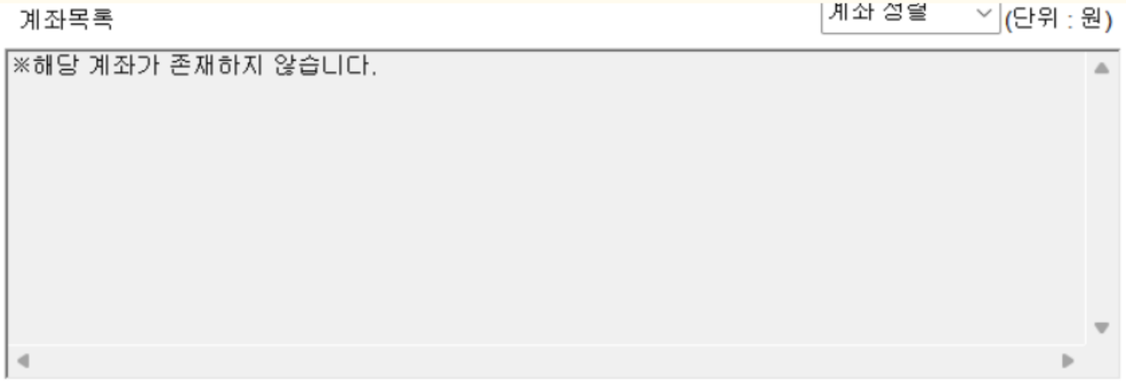
예금주 명을 이용한 계좌 검색 화면

1. 사용자가 알맞은 예금주명 데이터 입력
2. 검색 버튼을 누르면 사용자의 정보가 계좌목록에 출력
3. 찾는 예금주가 없다면 정보가 없다고 사용자에게 정보 전달




```
public Account findAccount(String accountNum) { return (Account)this.accounts.get(accountNum); }

public List<Account> findAccountByAccountOwner(String accountOwner) {
    List<Account> findAccounts = new ArrayList();
    Collection<Account> values = this.accounts.values();
    Iterator var4 = values.iterator();

    while(var4.hasNext()) {
        Account account = (Account)var4.next();
        String owner = account.getAccountOwner();
        if (owner.equals(accountOwner)) {
            findAccounts.add(account);
        }
    }

    return findAccounts;
```

AccountService클래스에서 배열로 받은
accounts를 roop돌면서 찾은 후 계좌목록에 출력



사용자가 검색한 계좌가 없다면 존재하지 않는다고 사용자에게 전달

프로젝트 후기

프로젝트 후기 -
JAVA를 처음 4주정도 배우고 난 후 진행한 프로젝트여서 시간도 오래걸리고 많이 어려워 했지만 그래도 내 손으로 만든 작은 프로젝트가 작동을 하는것을 보며 굉장히 뿌듯했고 성취감도 좋았습니다. 특히 처음 배울때는 main클래스에서 모든 코드를 작성을 하다보니 보기도 불편하고 비효율적이라고 생각했는데 class와 객체의 개념을 배워서 객체지향프로그래밍을 하려고 노력하니 너무 신기하고 편리했습니다.

좋았던점

1. JAVA를 한달 배우고 만든 작은 프로젝트지만 생각보다 잘 만든것 같아 뿌듯했다.
2. 최대한 객체지향 프로그램을 하려 노력했다.
3. 유효성 검증을 하는데 시간이 오래걸렸지만 잘 작동을 해서 뿌듯했다.
4. String의 format매서드를 잘 이해하기 좋은 프로젝트 였다.

아쉬웠던 점

1. 화면 구성을 할때 좌표값으로 하니 매우 불편하고 확인하기 힘들었다.
2. 내가 입력한 데이터값이 다시 켜면 없어지기 때문에 다시 등록해야하는 점이 불편했다.
3. 데이터 정렬 기능을 완벽하게 수행하지 못해서 아쉬었다.