C++

- the convenient ATM

Software Modeling & Analysis

OOPT Stage 2050 & 2060
Construct & Testing

T2.
201411262 김도현
201411271 박상우
201411312 장하나
201411316 정진호

INDEX

Activity 2051. Implement Class & Methods Definitions

Activity 2052. Implement Windows

Activity 2055. Write Unit Test Code

Activity 2061. Unit Testing

Activity 2063. System Testing

Activity 2067. Testing Traceability Analysis

Activity 2051. Implement Class & Method Definitions

Type	Class
Name	Account
Purpose	Account의 정보를 가지고 있다.
Overview	N/A
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R4.1, R4.2, R7, R8
Exc. Course of Events	N/A

Type	Method
Name	checkAccountPassword
Purpose	올바른 비밀번호를 입력했는지 확인한다.
Cross References	Functions : R2.1, R2.2, R3.1, R3.2, R4.1, R4.2, R8
Input	password : Integer
Output	validate : Boolean
Abstract operation	Input과 Account의 password를 비교해 결과를 반환한다.
Exc. Course of Events	N/A

Type	Method
Name	saveAccount
Purpose	Account를 저장한다.
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R7, R8
Input	void
Output	void
Abstract operation	DataStore를 이용하여 Account를 저장한다.
Exc. Course of Events	N/A

Type	Class
Name	Admin
Purpose	Admin의 정보를 가지고 있다.
Overview	N/A
Cross References	Functions : R10, R11, R12.1, R12.2
Exc. Course of Events	N/A

Type	Method
Name	checkAdminAccount
Purpose	올바른 계정 정보를 입력했는지 확인한다.
Cross References	Functions : R10, R11, R12.1, R12.2
Input	adminID : String, adminPW : String
Output	validate : Boolean
Abstract operation	Input과 Admin의 id, password를 비교해 결과를 반환한다.
Exc. Course of Events	N/A

Type	Class
Name	DataStore
Purpose	Account와 User의 정보를 불러오거나 저장한다.
Overview	N/A
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R4.1, R4.2, R6, R7, R8
Exc. Course of Events	N/A

Type	Method
Name	loadAccountData
Purpose	Account를 불러온다.
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R4.1, R4.2, R7, R8
Input	accountNo : String
Output	account : Account
Abstract operation	Input에 해당하는 Account를 불러와 Account 객체를 생성하고 반환하다.
Exc. Course of Events	N/A

Type	Class
Name	Lottery
Purpose	Lottery의 정보를 가지고 있다.
Overview	N/A
Cross References	Functions : R7
Exc. Course of Events	N/A

Type	Method

Name	checkResult
Purpose	Lottery의 당첨금을 확인한다.
Cross References	Functions : R7
Input	void
Output	prizeAmount : Integer
Abstract operation	Lottery의 week와 numbers를 보내 등수를 받아오고, 해당하는 당첨금을 반환한다.
Exc. Course of Events	N/A

Type	Class
Name	ATMSystem
Purpose	프로그램의 수행을 제어한다.
Overview	N/A
Cross References	Functions : All
Exc. Course of Events	N/A

Type	Method
Name	getCurrency
Purpose	원-딜러 환율을 받아온다.
Cross References	Functions : R1.2, R2.2
Input	void
Output	currency : Double
Abstract operation	원-달러 환율을 받아와 반환한다.
Exc. Course of Events	N/A

Type	Method
Name	requestStopCard
Purpose	카드 정지를 요청한다.
Cross References	Functions : R6
Input	cardNumber : String
Output	void
Abstract operation	(실제 카드 정지를 요청할 수 없으므로) 카드 정지 메세지를 출력한다.
Exc. Course of Events	N/A

Type	Method
Name	createAdminID
Purpose	Admin의 id를 자동 생성한다.
Cross References	Functions : R10
Input	void
Output	newId : String
Abstract operation	마지막 관리자의 id에서 1을 더한 값을 반환한다.
Exc. Course of Events	N/A

Type	Class
Name	SystemBalance
Purpose	ATM의 지폐 보유량과 지폐 보유량의 제한값을 관리한다.
Overview	N/A
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R12.1, R12.2
Exc. Course of Events	N/A

Type	Class
Name	SystemState
Purpose	ATM의 상태와 언어를 관리한다.
Overview	N/A
Cross References	Functions : R9, R11
Exc. Course of Events	N/A

Type	Method
Name	toggleState
Purpose	ATM의 사용자 기능 상태를 전환한다.
Cross References	Functions : R11
Input	void
Output	void
Abstract operation	SystemState의 state를 전환한다.
Exc. Course of Events	N/A

Type	Class
Name	Transaction

Purpose	Transaction의 정보를 가지고 있다.
Overview	N/A
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R7
Exc. Course of Events	N/A

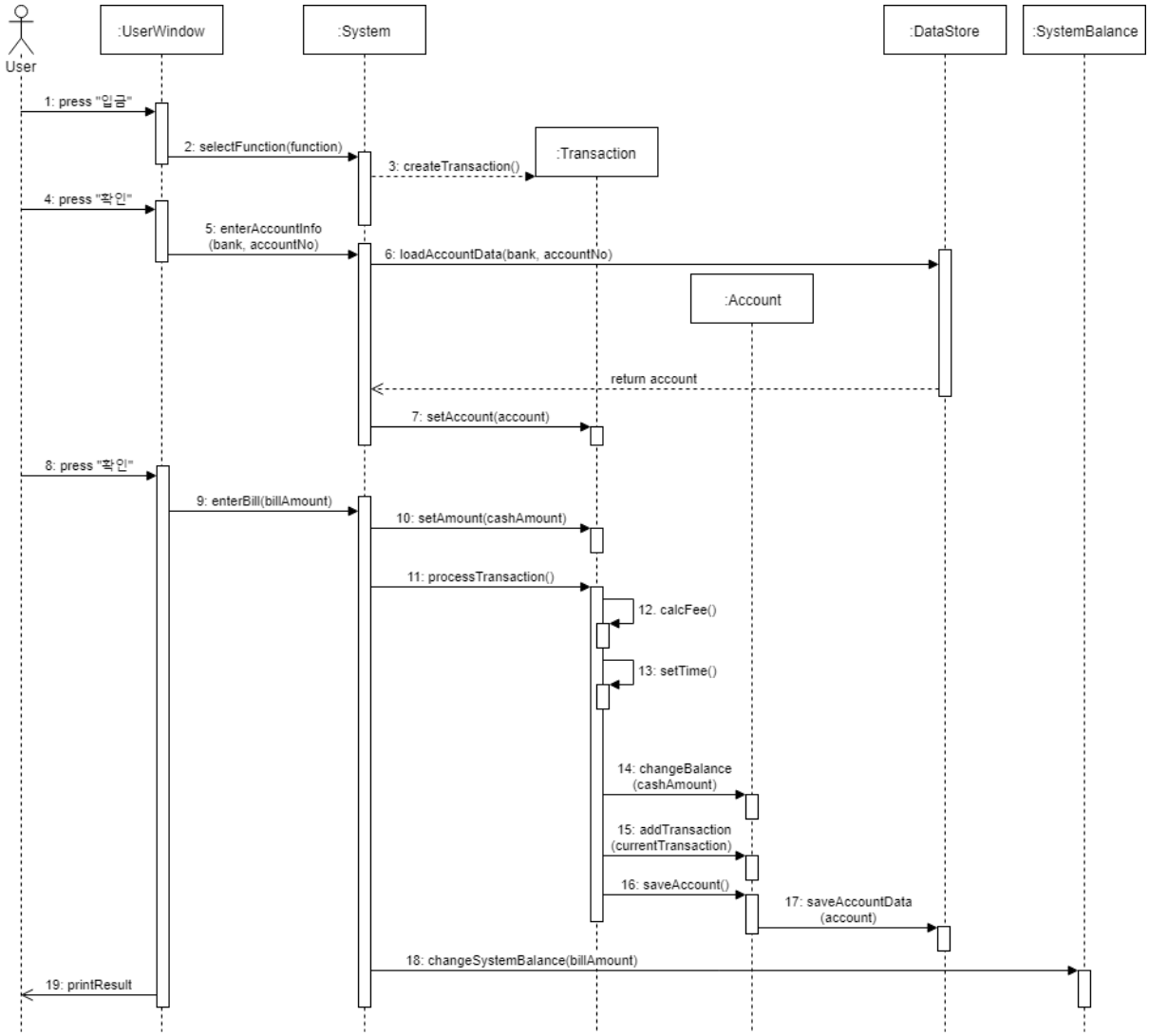
Type	Method
Name	processTransaction
Purpose	거래를 진행한다.
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R7
Input	void
Output	void
Abstract operation	수수료 계산, 거래 시간 기록, 계좌 잔액 변경, 거래 기록 추가, 계좌 저장을 순차적으로 진행한다.
Exc. Course of Events	N/A

Type	Method
Name	calcFee
Purpose	수수료를 계산한다.
Cross References	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R5, R7
Input	void
Output	void
Abstract operation	거래의 종류와 은행을 참조하여 수수료를 계산한 후, Transaction의 amount를 변경한다.
Exc. Course of Events	N/A

Type	Class
Name	User
Purpose	User의 정보를 가지고 있다.
Overview	N/A
Cross References	Functions : R6
Exc. Course of Events	N/A

Activity 2052. Implement Windows

1. Deposit



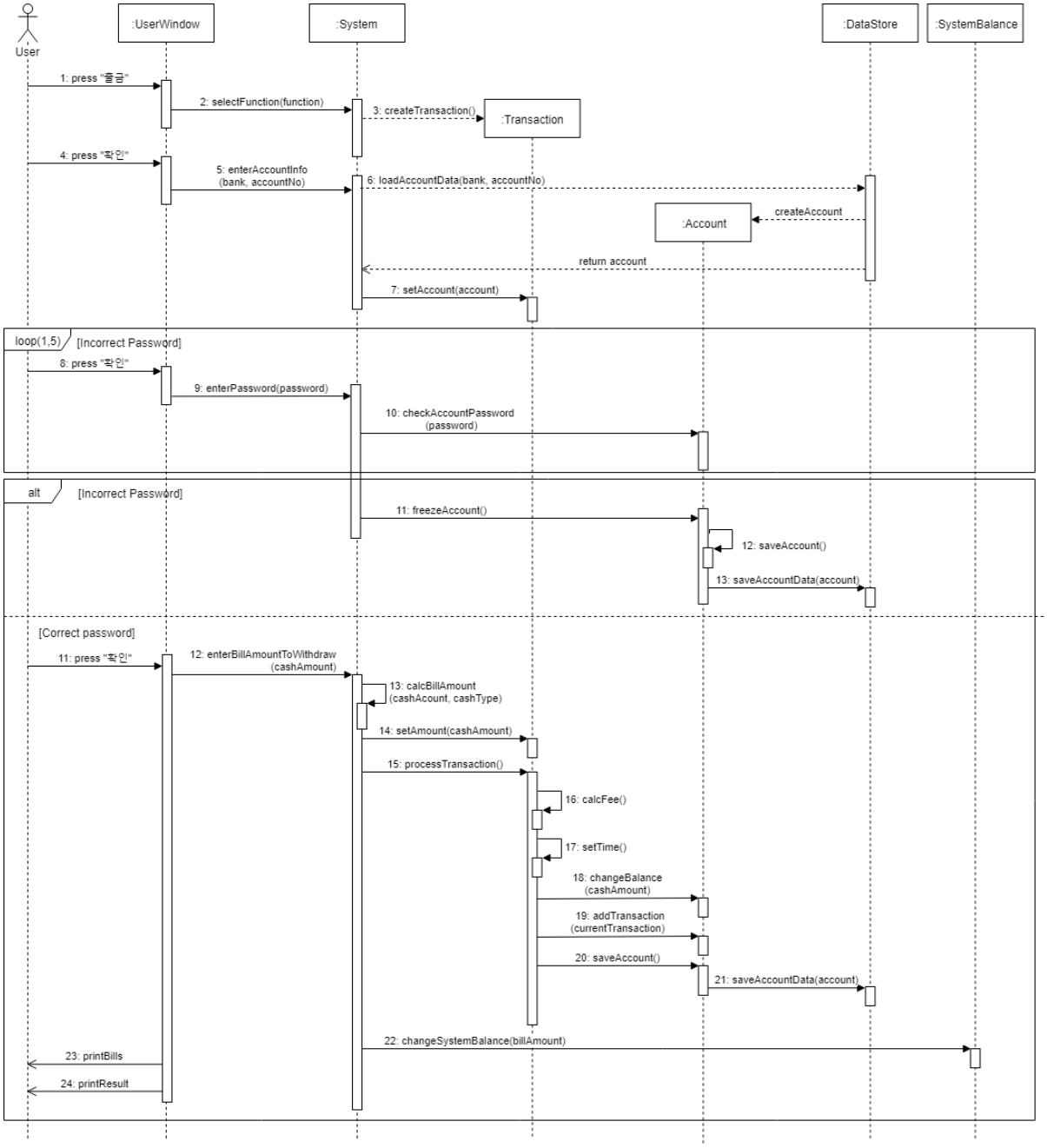
Name	1. press "입금"
Responsibilities	메뉴 선택 화면의 "입금" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R1.1,
Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

Name	4. press "확인"
Responsibilities	계좌 정보를 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R4.1, R4.2, R7
Notes	N/A
Pre-Conditions	은행과 계좌 번호가 모두 입력되어 있어야 한다.
Post-Conditions	N/A

Name	8. press "확인"
Responsibilities	입금할 지폐를 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R1.1, R1.2
Notes	N/A
Pre-Conditions	지폐가 입력되어 있어야 한다.
Post-Conditions	N/A

Name	19. printResult
Responsibilities	거래 결과를 출력한다
Type	GUI
Cross Reference	Functions : R1.1, R1.2, R2.1, R2.2, R3.1, R3.2, R7
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

2. Withdraw



Name	1. press "출금"
Responsibilities	메뉴 선택 화면의 "출금" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R2.1
Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.

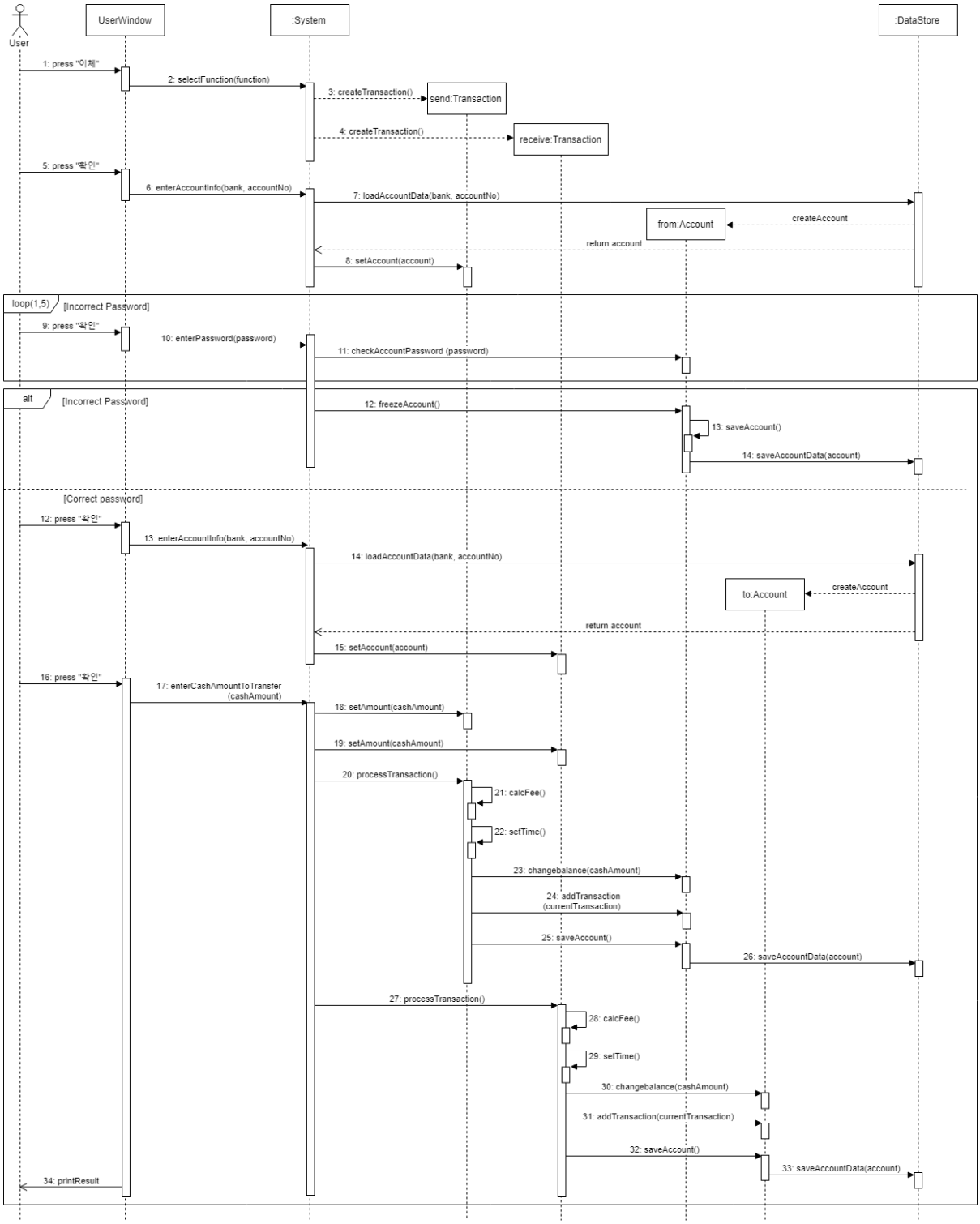
Post-Conditions	N/A

Name	8. press "확인"
Responsibilities	비밀번호를 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R2.1, R2.2, R3.1, R3.2, R4.1, R4.2,
Notes	N/A
Pre-Conditions	비밀번호가 입력되어 있어야 한다.
Post-Conditions	N/A

Name	11. press "확인"
Responsibilities	출금할 금액을 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R2.1, R2.2
Notes	N/A
Pre-Conditions	금액이 입력되어 있어야 한다.
Post-Conditions	N/A

Name	23. printBills
Responsibilities	지폐를 출력한다.
Type	GUI
Cross Reference	Functions : R2.1, R2.2
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

3. Transfer Balance

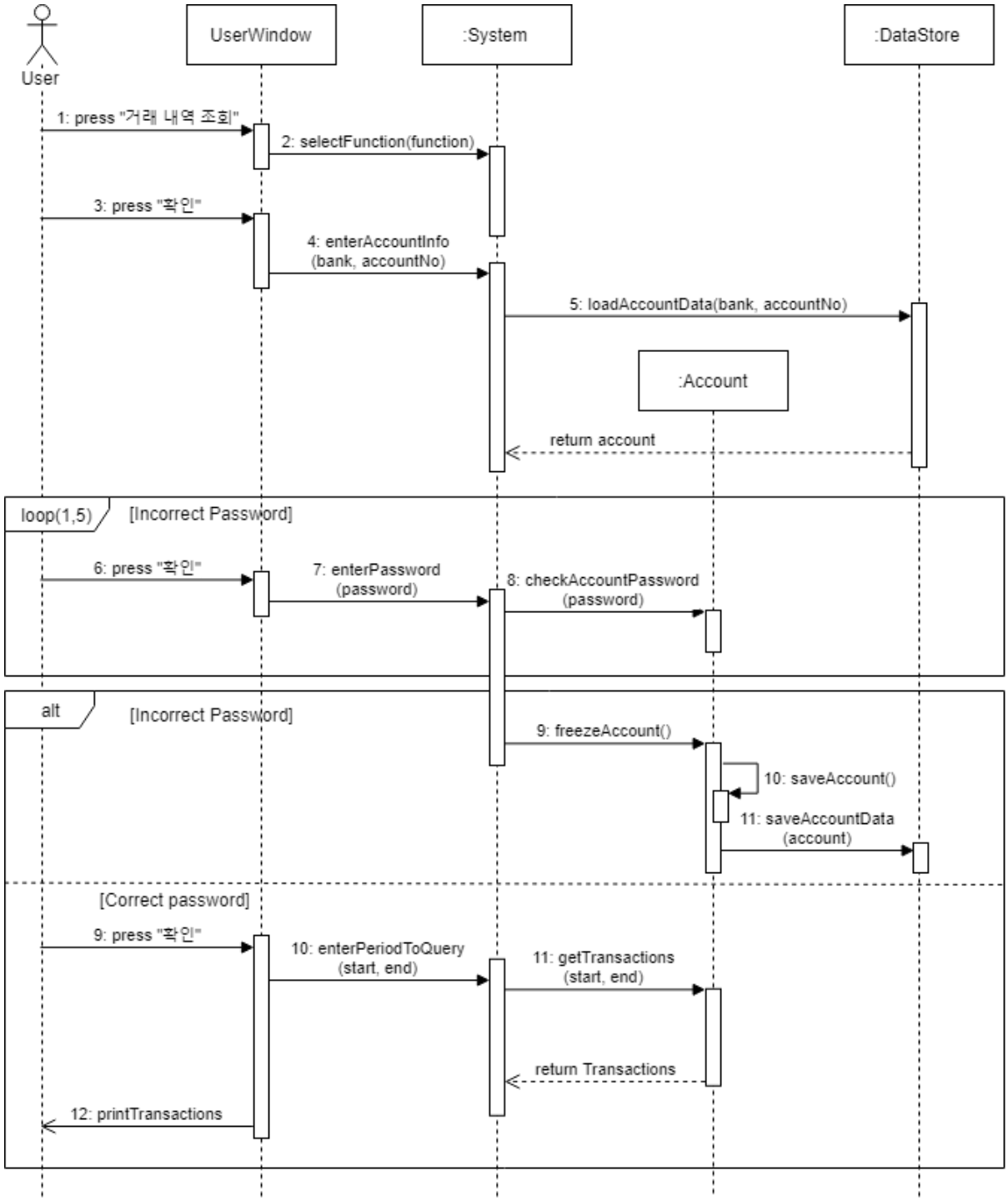


Name	1. press "이체"
Responsibilities	메뉴 선택 화면의 "이체" 버튼을 누른다.

Type	GUI
Cross Reference	Functions : R3.1
Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

Name	16. press "확인"
Responsibilities	이체할 금액을 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R3.1, R3.2
Notes	N/A
Pre-Conditions	금액이 입력되어 있어야 한다.
Post-Conditions	N/A

4. Query Transaction List



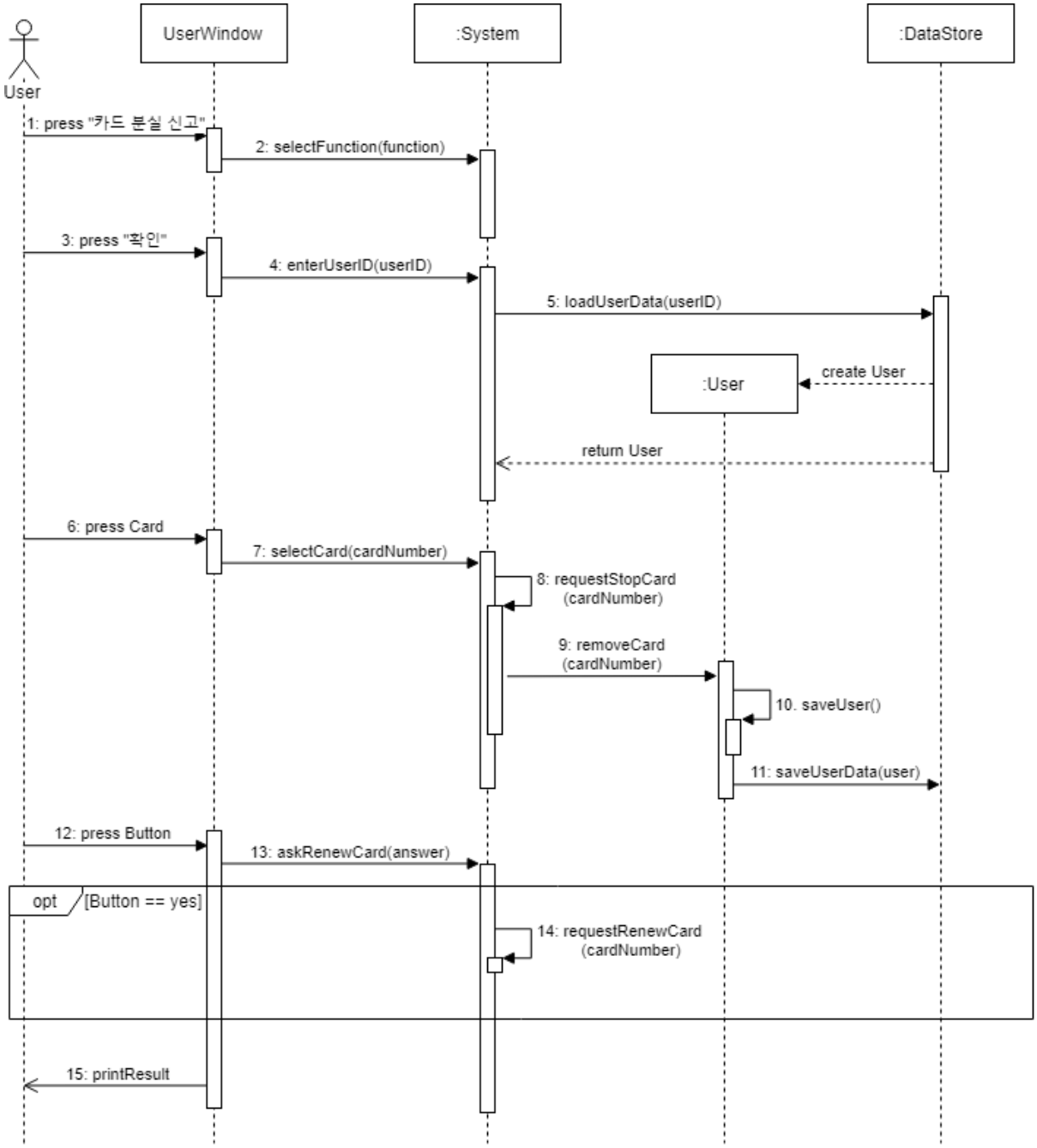
Name	1. press "거래 내역 조회"
Responsibilities	메뉴 선택 화면의 "거래 내역 조회" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R4.1

Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

Name	9. press "확인"
Responsibilities	거래 내역을 조회할 기간을 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R4.1
Notes	N/A
Pre-Conditions	시작 일자와 종료 일자가 모두 입력되어 있어야 한다.
Post-Conditions	N/A

Name	12. printTransactions
Responsibilities	거래 내역을 출력한다.
Type	GUI
Cross Reference	Functions : R4.1
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

5. Report Lost Card



Name	1. press "카드 분실 신고"
Responsibilities	메뉴 선택 화면의 "카드 분실 신고" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R6
Notes	N/A

Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

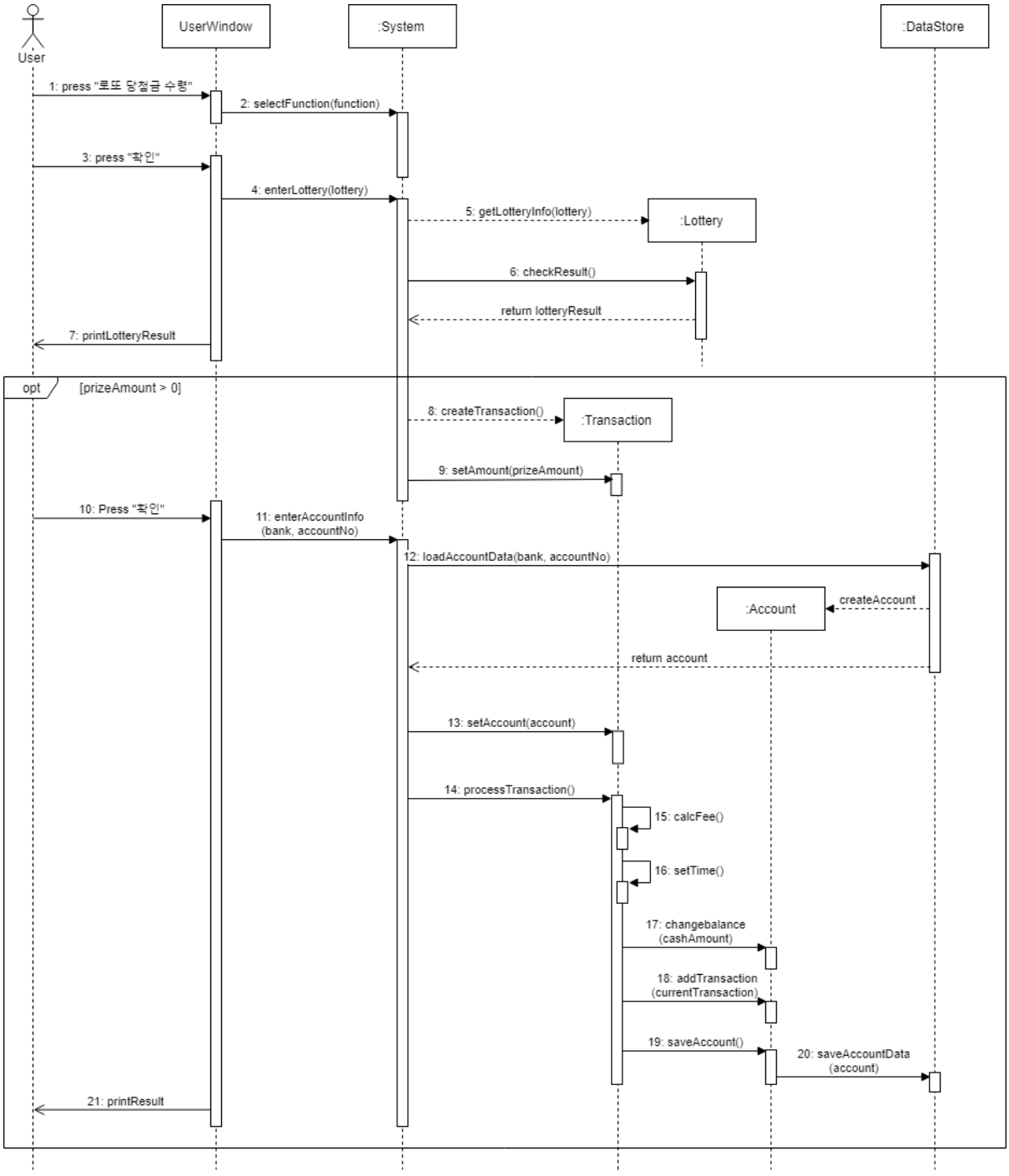
Name	3. press "확인"
Responsibilities	개인식별번호를 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R6
Notes	N/A
Pre-Conditions	개인식별번호가 입력되어 있어야 한다.
Post-Conditions	N/A

Name	6. press Card
Responsibilities	정지할 카드를 선택한다.
Type	GUI
Cross Reference	Functions : R6
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	12. press Button
Responsibilities	카드 재발급 여부를 선택한다.
Type	GUI
Cross Reference	Functions : R6
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	15. printResult
Responsibilities	분실 신고 (및 재발급 신청) 결과를 출력한다.
Type	GUI
Cross Reference	Functions : R6
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

6. Get Lottery Prize



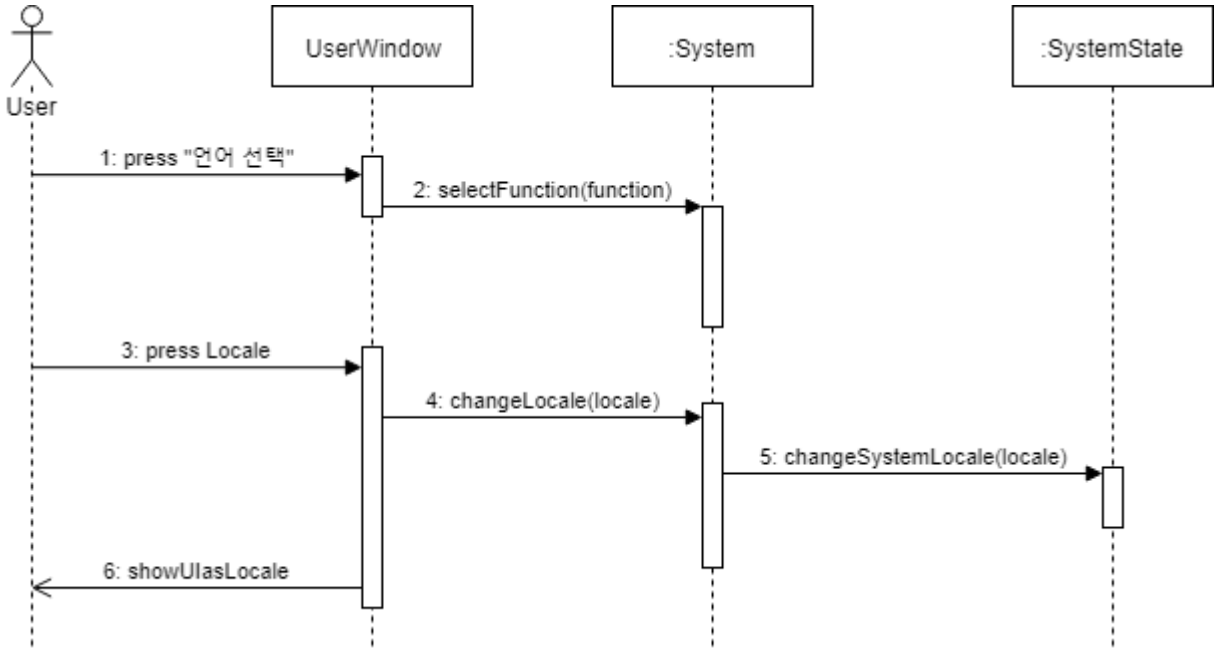
Name	1. press "로또 당첨금 수령"
Responsibilities	메뉴 선택 화면의 "로또 당첨금 수령" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R7

Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

Name	3. press "확인"
Responsibilities	로또 정보를 입력하고 "확인" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R7
Notes	N/A
Pre-Conditions	주차와 번호가 모두 입력되어 있어야 한다.
Post-Conditions	N/A

Name	7. printLotteryResult
Responsibilities	로또 결과를 출력한다.
Type	GUI
Cross Reference	Functions : R7
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

7. Change Locale



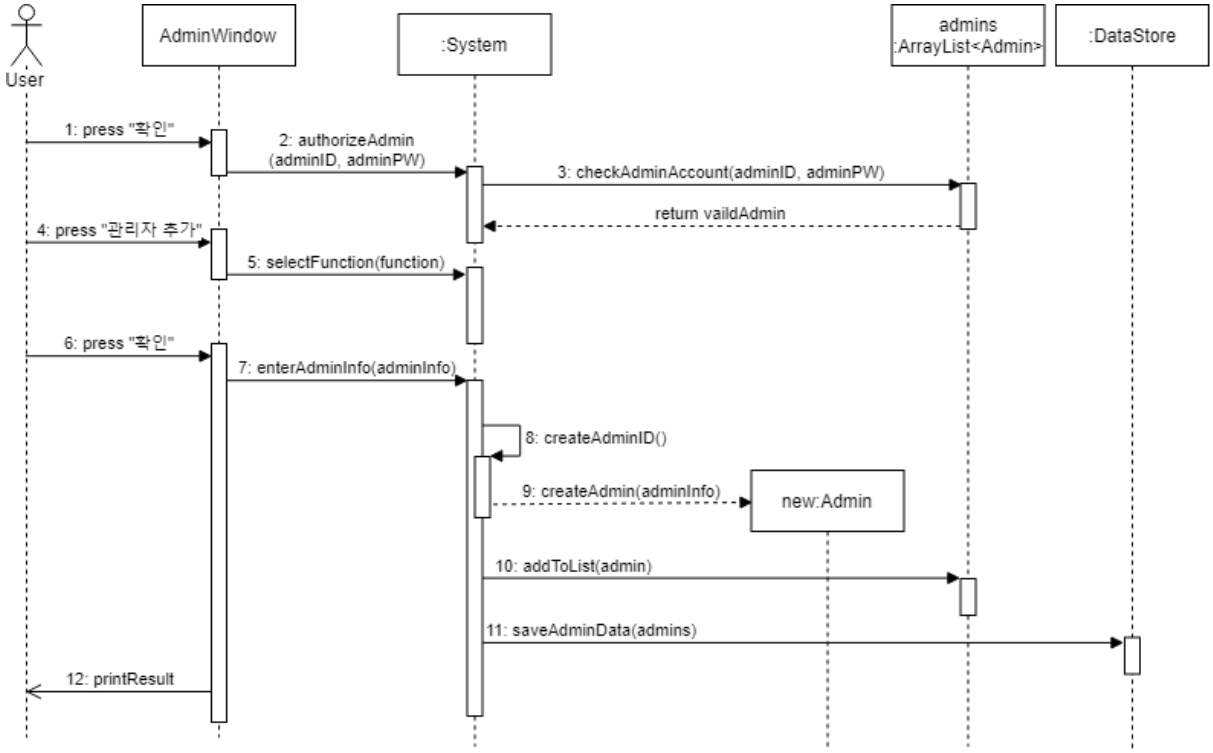
Name	1. press "언어 선택"
Responsibilities	메뉴 선택 화면의 "언어 선택" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R9
Notes	N/A
Pre-Conditions	ATM이 사용자 제한 상태가 아니다.
Post-Conditions	N/A

Name	3. press Locale
Responsibilities	언어를 선택한다.
Type	GUI
Cross Reference	Functions : R9
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	6. showUIasLocale
Responsibilities	선택한 언어로 UI를 재출력한다.
Type	GUI

Cross Reference	Functions : R9
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

8. Add Admin



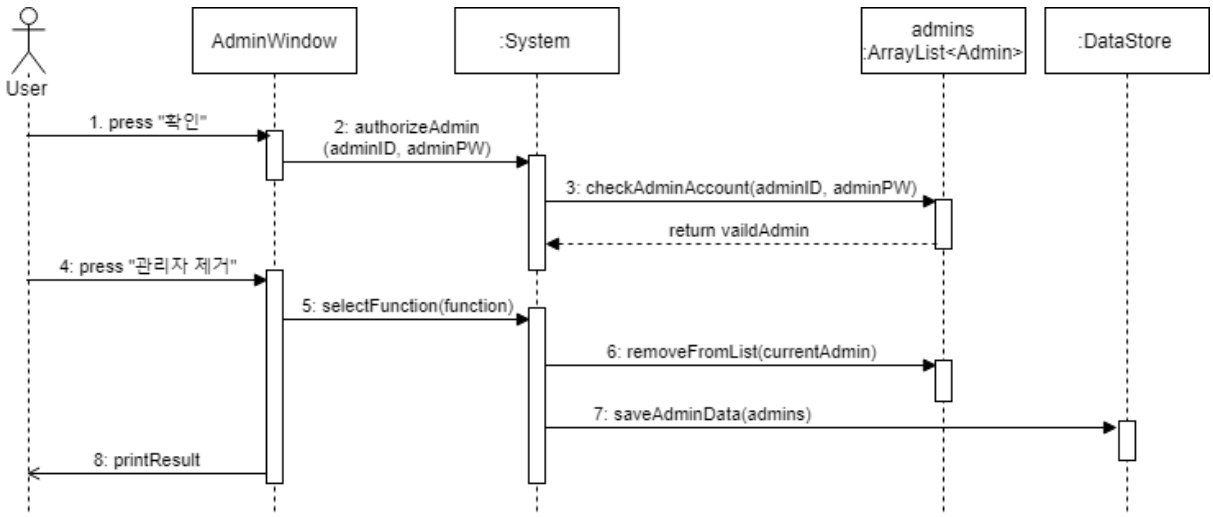
Name	1. press "확인"
Responsibilities	관리자 계정 정보를 입력한다.
Type	GUI
Cross Reference	Functions : R10, R11, R12.1, R12.2
Notes	N/A
Pre-Conditions	아이디와 패스워드 모두 입력되어 있어야 한다.
Post-Conditions	N/A

Name	4. press "관리자 추가"
Responsibilities	메뉴 선택 화면의 "관리자 추가" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R10
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	6. press "확인"
Responsibilities	새로운 관리자 계정 정보를 입력한다.
Type	GUI
Cross Reference	Functions : R10
Notes	N/A
Pre-Conditions	password와 contact 모두 입력되어 있어야 한다.
Post-Conditions	N/A

Name	12. printResult
Responsibilities	관리자 추가 결과를 출력한다.
Type	GUI
Cross Reference	Functions : R10
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

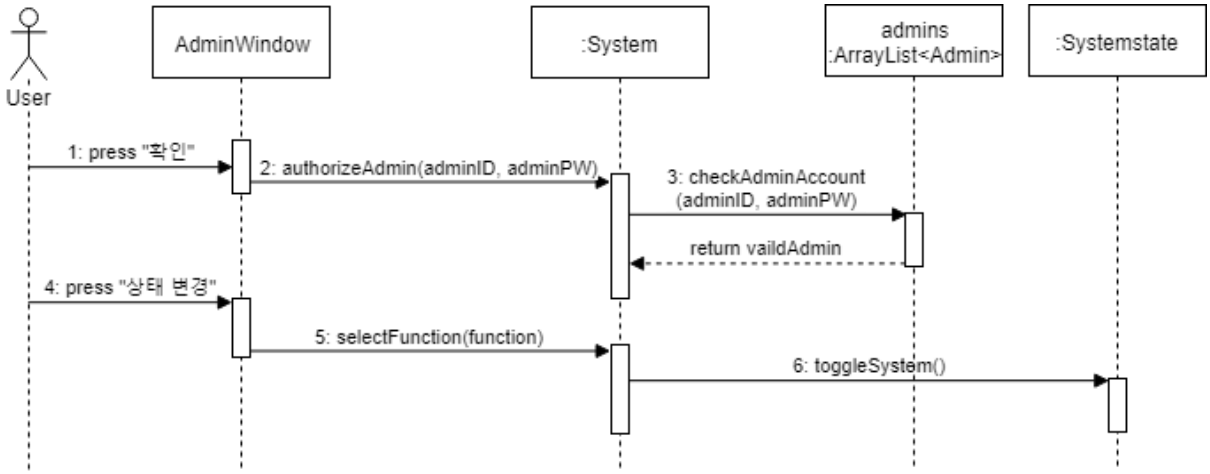
9. Remove Admin



Name	4. press "관리자 제거"
Responsibilities	메뉴 선택 화면의 "관리자 제거" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R10
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

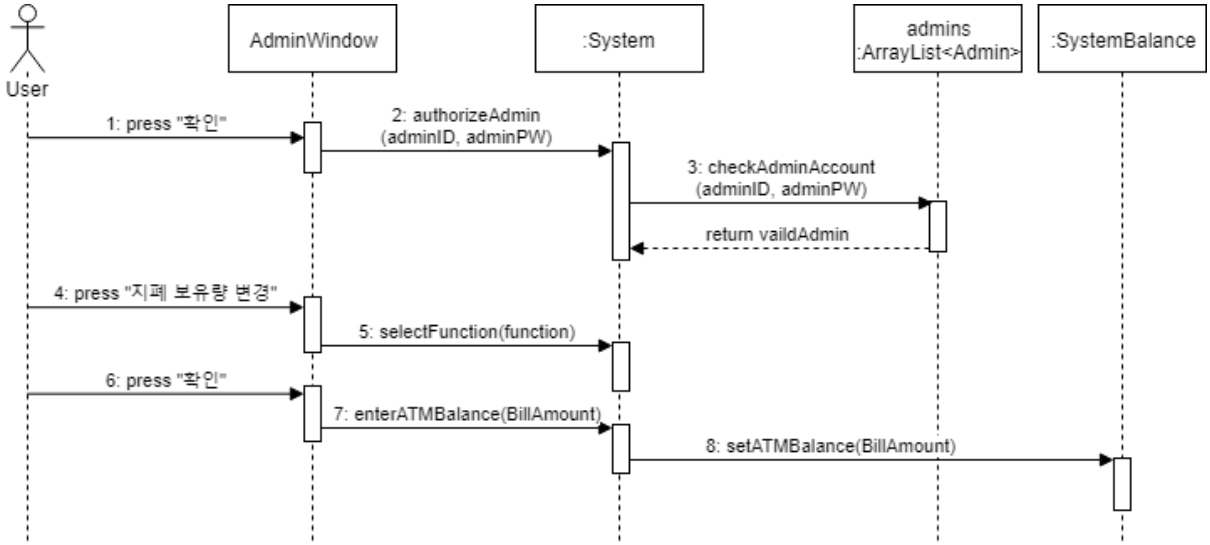
Name	8. printResult
Responsibilities	관리자 삭제 결과를 출력한다.
Type	GUI
Cross Reference	Functions : R10
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

10. Toggle ATM State



Name	4. press "상태 변경"
Responsibilities	메뉴 선택 화면의 "상태 변경" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R11
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

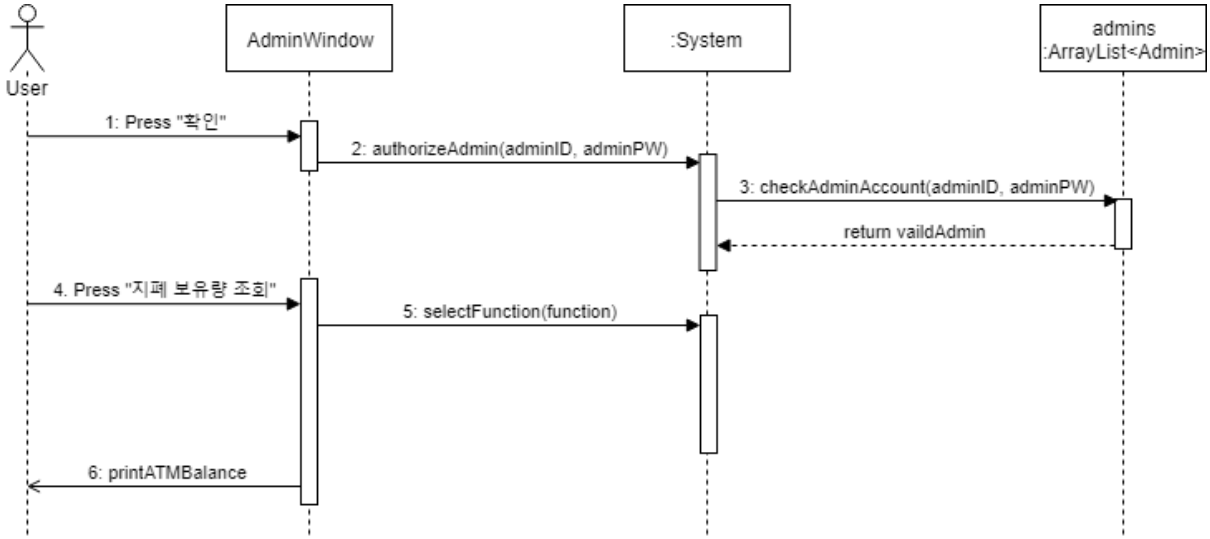
11. Change ATM Balance



Name	4. press "지폐 보유량 변경"
Responsibilities	메뉴 선택 화면의 "지폐 보유량 변경" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R12.1
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	4. press "확인"
Responsibilities	변경할 지폐 보유량 값을 입력한다.
Type	GUI
Cross Reference	Functions : R12.1
Notes	N/A
Pre-Conditions	지폐 종류 별로 모두 입력되어 있어야 한다.
Post-Conditions	N/A

12. Query ATM Balance



Name	4. press "지폐 보유량 조회"
Responsibilities	메뉴 선택 화면의 "지폐 보유량 조회" 버튼을 누른다.
Type	GUI
Cross Reference	Functions : R12.2
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Name	7. printATMBalance
Responsibilities	지폐 종류 별 보유량을 출력한다.
Type	GUI
Cross Reference	Functions : R12.2
Notes	N/A
Pre-Conditions	N/A
Post-Conditions	N/A

Activity 2055. Write Unit Test Code

1. ATMSystem Test

```
public class ATMSystemTest {
    public ATMSystem system;

    @Before
    public void initATMSystem() throws Exception {
        this.system = new ATMSystem();
        this.system.selectFunction(FunctionType.Deposit);
}

    @Test(expected = InvalidBillException.class)
    public void enterBillRaisesExceptionIfLengthIsInvalid() throws InvalidBillException, DataStoreError, OverflowBillException {
        int[] billAmount = {1, 2};
        system.enterBill(billAmount);
    }

    @Test(expected = InvalidBillException.class)
    public void enterBillAsDollarRaisesExceptionIfLengthIsInvalid() throws InvalidBillException, DataStoreError, OverflowBillException {
        int[] billAmount = {1, 2};

        system.enterBillAsDollar(billAmount);
    }

    @Test(expected = AccountDoesNotExist.class)
    public void enterAccountInfoRaisesExceptionIfAccountDoesNotExistOnDataStore()
        throws AccountDoesNotExist, FrozenAccountException, DataStoreError, NoneOfFunctionSelected {
        system.enterAccountInfo(Bank.WOORI, "DOESNOTEXIST");
    }
    @Test
    public void enterBillAmountToWithdrawAsDollarCorrectlyWorking() {
        try {
            system.selectFunction(FunctionType.ForeignWithdraw);
        } catch (NoneOfFunctionSelected e) {

        }

        try {
            system.enterAccountInfo(Bank.HANA, "123456789012345");
        } catch (Exception e) {

        }

        int []initialAmount = {500,500,500,500,500,500,500,500,500,500,500};

        try {
            system.getBalance().setATMBalance(initialAmount);
        } catch (OverflowBillException ex) {

        }

        try {
            system.enterBillAmountToWithdrawAsDollar(-200);
        } catch (DataStoreError | NegativeBalanceError | OverflowBillException e) {
            fail(e.getClass().getSimpleName());
        }

        assertEquals(500-2, system.getBalance().getATMBalance()[10]);
    }
```

```
@Test
public void enterAccountInfoChangesProperty() {
    try {
        system.enterAccountInfo(Bank.HANA, "123456789012345");
    } catch (AccountDoesNotExist | DataStoreError e) {
        fail(e.getClass().getSimpleName());
    } catch (NoneOfFunctionSelected e) {

    } catch (FrozenAccountException e) {

    }

    assertNotNull(system.getAccount());
    assertEquals(system.getAccount().getBank(), Bank.HANA);
    assertEquals(system.getAccount().getAccountNo(), "123456789012345");
}

@Test
public void enterPasswordSuccess() {
    try {
        system.enterAccountInfo(Bank.HANA, "123456789012345");
    } catch (AccountDoesNotExist e) {
        fail("throw AccountDoesNotExist " + e.getMessage());
    } catch (DataStoreError e) {
        fail("throw DataStoreError " + e.getMessage());
    } catch (NoneOfFunctionSelected e) {

    } catch (FrozenAccountException e) {
        fail("throw FrozenAccountError" + e.getMessage());
    }

    try {
        system.enterPassword(5555);
    } catch (InvalidPasswordException e) {
        fail("throw InvalidPasswordException " + e.getMessage());
    } catch (AccountDoesNotExist e) {
        fail("throw AccountDoesNotExist" + e.getMessage());
    } catch(DataStoreError | NegativeBalanceError e){

    } catch (FrozenAccountException e) {
        fail("Account is Frozen");
    }

    // PASS
}
```

```
@Test(expected = FrozenAccountException.class)
public void enterPasswordFreezesAccountWhenInvalidInputIsRepeatForFiveTimes() throws FrozenAccountException {
    try {
        system.enterAccountInfo(Bank.HANA, "123456789012345t");
    } catch (AccountDoesNotExist e) {
        fail("throw AccountDoesNotExist" + e.getMessage());
    } catch (DataStoreError e) {
        fail("throw DataStoreError" + e.getMessage());
    } catch (NoneOfFunctionSelected e) {

    } catch (FrozenAccountException e) {

    }

    while(true){
        System.out.println("Input Password");
        try {
            system.enterPassword(0000);
        } catch (AccountDoesNotExist e) {
            fail("throw AccountDoesNotExist" + e.getMessage());
        } catch (DataStoreError | NegativeBalanceError e) {
            fail("throw Error");
        } catch (InvalidPasswordException e){

        }
    }
}

@Test
public void calcBillAccountCorrectlyWorking() {
    int[] result = system.calcBillAmount(100000, "WON");
    int[] expectedResult = {0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, };

    assertArrayEquals(expectedResult, result);

    result = system.calcBillAmount(130, "Dollar");
    expectedResult = new int[]{0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1};

    assertArrayEquals(expectedResult, result);
}

@Test(expected = TooFewUser.class)
public void enterNumberOfUsersInputIsZero() throws TooFewUser {
    system.enterNumberOfUsers(0);
}
@Test
public void enterNumberOfUsersSuccessfullyDivideCashAmount() throws NoneOfFunctionSelected {
    system.selectFunction(FunctionType.SplitPay);
    system.setCashAmount(1000);

    try {
        system.enterNumberOfUsers(5);
    } catch (TooFewUser e) {
        fail("throw TooFewUser" + e.getMessage());
    }

    assertEquals(system.getCashAmount(), 1000/5);


}
```

```
@Test
public void enterLotteryMakesTransaction() {
    int[] winningNumbers = {6, 10, 18, 25, 34, 35};
    Lottery lottery = new Lottery(807, winningNumbers);

    try {
        system.enterLottery(lottery);
    } catch (LotteryFailed e) {
        fail("throw LotteryFailed" + e.getMessage());
    }

    assertEquals(50000000, system.getToTransaction().getAmount());
}

@Test(expected = LotteryFailed.class)
public void enterLotteryFailedWhenLose() throws LotteryFailed {
    int[] loseNumbers = {1, 2, 3, 4, 5, 7};
    Lottery lottery = new Lottery(807, loseNumbers);

    system.enterLottery(lottery);
}

@Test
public void enterAdminInfoCorrectlyWorking() {
    try {
        system.enterAdminInfo("1234", "01012341234");
    } catch (DataStoreError er) {

    }
    Admin[] admins = system.getAdmins();

    assertEquals("1234", admins[admins.length - 1].getPassword());
    assertEquals("01012341234", admins[admins.length - 1].getContact());

    try {
        system.authorizeAdmin(admins[admins.length - 1].getId(), "1234");
        system.selectFunction(FunctionType.RemoveAdmin);
    } catch (InvalidAdminException ex) {
        fail("DeleteFailed");
    } catch (NoneOfFunctionSelected ex) {

    }
}
```

```
    @Test
    public void shouldGetCardListCorrectly() throws UserDoestNotExist {
        this.system.enterUserId("123456789012356");
        String[] string;
        string = this.system.getUser().getCardList();
        assertTrue(string[0].equals("123456789012345678"));
    }

    @Test
    public void enterPeriodToQueryDoesGetTransactions() throws NoneOfFunctionSelected {
        system.selectFunction(FunctionType.QueryTransactionList);

        try {
            system.enterAccountInfo(Bank.HANA, "123456789012345");
            system.enterPeriodToQuery(new Date(2016-1900,5-1,1),
                new Date(2018-1900,5-1,30));
        } catch (AccountDoesNotExist | DataStoreError ex) {

        } catch (FrozenAccountException e) {

        }
        assertTrue(system.getTransactionList().length > 0);
    }

    @After
    public void restoreFile() throws DataStoreError {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        account.saveAccount();
    }
}
```

2. Account Test

```
public class AccountTest {
    private DataStore dataStore;

    @Before
    public void initDataStore() {
        this.dataStore = new DataStore();
    }

    @Test
    public void shouldGetCorrectAccount() {
        Account account = this.dataStore.loadAccountData(Bank.HANA, "123456789012345t");
        assertEquals(account.getBalance(), 1004200);
        assertEquals(account.getPassword(), 5555);
    }

    @Test
    public void shouldGetCorrectBalance() {
        Account account = this.dataStore.loadAccountData(Bank.HANA, "123456789012345t");
        try {
            account.changeBalance(5000);
        } catch (NegativeBalanceError e) {
            fail(e.getClass().getSimpleName());
        }
        assertEquals(account.getBalance(), 1009200);
    }

    @Test
    public void shouldGetCorrectTransactions() {
        Account account = this.dataStore.loadAccountData(Bank.HANA, "123456789012345t");
        ArrayList<Transaction> transactions = account.getTransactions(new Date(118, 4, 19, 00, 0, 0), new Date(118, 4, 20, 0, 0, 0));
        assertEquals(transactions.get(0).getAmount(), 1400);
        assertEquals(transactions.get(0).getTime(), new Date(118, 4, 19, 15, 16, 59));
        assertEquals(transactions.size(), 1);
    }

    @Test
    public void shouldAddTransactionCorrectly() {
        Account account = new Account(Bank.KOOKMIN, "15151515141414");
        Date today = new Date();
        Date startToday = new Date(today.getYear(), today.getMonth(), today.getDate());
        Date endToday = new Date(today.getYear(), today.getMonth(), today.getDate()+1);

        Transaction transaction = new Transaction(TransactionType.Deposit);
        transaction.setAccount(account);
        transaction.setAmount(5100);
        transaction.setTime();
        account.addTransaction(transaction);

        ArrayList<Transaction> transactions = account.getTransactions(startToday, endToday);
        assertEquals(transactions.get(0).getAmount(), 5100);
    }
```

```
    @Test
    public void checkPasswordTest() {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        assertTrue(!account.checkAccountPassword(1234));
        assertTrue(account.checkAccountPassword(5555));
    }

    @Test
    public void shouldFreezeAccountState() {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        assertTrue(account.getState());
        account.freezeAccount();
        assertFalse(account.getState());
        account.freezeAccount();
        assertFalse(account.getState());
    }

    @Test
    public void shouldChangeBalanceCorrectly() throws NegativeBalanceError {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        assertEquals(account.getBalance(), 1004200);
        account.changeBalance(10000000);
        assertEquals(account.getBalance(), 10000000+1004200);
        account.changeBalance(-5000000);
        assertEquals(account.getBalance(), 10000000+1004200-5000000);
    }

    @Test
    public void shouldSaveAccountCorrectly() throws NegativeBalanceError, DataStoreError {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        account.changeBalance(10000000);
        account.saveAccount();
        Account account2 = dataStore.loadAccountData(Bank.HANA, "123456789012345");
        assertEquals(account2.getBalance(), 10000000+1004200);
    }

    @After
    public void restoreFile() throws DataStoreError {
        Account account = dataStore.loadAccountData(Bank.HANA, "123456789012345t");
        dataStore.saveAccountData(account);
    }
}
```

3. Admin Test

```
public class AdminTest {

    @Test
    public void shouldCheckAdminAccount() {
        DataStore ds = new DataStore();
        ArrayList<Admin> adminList = ds.loadAdminData();
        assertTrue(adminList.get(0).checkAdminAccount("1", "1234"));
        assertFalse(adminList.get(0).checkAdminAccount("1", "1444"));
    }
}
```

4. DataStore Test

```
public class DataStoreTest {

    public DataStore dataStore;

    @Before
    public void init() {
        dataStore = new DataStore();
    }


    @Test
    public void loadAccountDataReturnNullWhenAccountIsInvalid() {
        Account account = dataStore.loadAccountData(Bank.HANA, "1234567890");
        assertNull(account);
    }

    @Test
    public void saveAccountDataCreatesFile() throws DataStoreError {
        Account account = new Account(Bank.KOOKMIN, "1010101010101010");
        account.saveAccount();

        File file = new File("data/" + account.getBank() + "/" + account.getAccountNo() + ".json");
        assertTrue(file.exists());
        file.delete();
    }

    @Test
    public void loadUserDataReturnNullWhenUserIsInvalid() {
        User user = new DataStore().loadUserData("1234567890");
        assertNull(user);
    }
}
```

5. Lottery Test

```
public class LotteryTest {

    @Test
    public void checkResultTest() {
        int[] num = {1, 2, 3, 4, 5, 7};
        Lottery lot = new Lottery(807, num);
        assertEquals(lot.checkResult(), 0);

        int[] num1 = {6, 10, 18, 4, 5, 7};
        lot = new Lottery(807, num1);
        assertEquals(lot.checkResult(), 5000);

        int[] num2 = {6, 10, 18, 25, 5, 7};
        lot = new Lottery(807, num2);
        assertEquals(lot.checkResult(), 50000);

        int[] num3 = {6, 10, 18, 25, 34, 7};
        lot = new Lottery(807, num3);
        assertEquals(lot.checkResult(), 500000);

        int[] num4 = {6, 10, 18, 25, 34, 33};
        lot = new Lottery(807, num4);
        assertEquals(lot.checkResult(), 5000000);

        int[] num5 = {6, 10, 18, 25, 34, 35};
        lot = new Lottery(807, num5);
        assertEquals(lot.checkResult(), 50000000);
    }
}
```

6. SystemBalance Test

```
public class SystemBalanceTest {
    @Test(expected = AdminAlarmException.class)
    public void throwAdminAlarmExceptionTest() throws AdminAlarmException, OverflowBillException {
        int[] balance = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50};
        int[] input = {40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        SystemBalance systemBalance = new SystemBalance(100, 80, 20, 0);
        systemBalance.setATMBalance(balance);
        systemBalance.changeSystemBalance(input);
    }

    @Test(expected = OverflowBillException.class)
    public void throwOverflowBillExceptionTest() throws AdminAlarmException, OverflowBillException {
        int[] balance = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50};
        int[] input = {60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        SystemBalance systemBalance = new SystemBalance(100, 80, 20, 0);
        systemBalance.setATMBalance(balance);
        systemBalance.changeSystemBalance(input);
    }
}
```

7. Transaction Test

```
public class TransactionTest {
    private DataStore dataStore;

    @Before
    public void initDataStore() {
        this.dataStore = new DataStore();
    }

    @Test
    public void calcFeeTest() {
        Account account = new DataStore().loadAccountData(Bank.WOORI, "110412644105");
        Transaction transaction = new Transaction(TransactionType.Deposit);
        transaction.setAccount(account);
        transaction.setAmount(10000);
        transaction.calcFee();
        assertEquals(transaction.getAmount(), 9000);
    }

    @Test
    public void processTransactionTest() {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345t");
        Transaction transaction = new Transaction(TransactionType.Deposit);
        transaction.setAccount(account);
        transaction.setAmount(10000);
        try {
            transaction.processTransaction();
        } catch (Exception e) {
            fail("throw Exception");
        }

        assertEquals(account.getTransactions(new Date(118, 4, 20, 0, 0, 0), new Date(118, 4, 21, 23, 0, 0)).size(), 0);
        assertEquals(account.getBalance(), 1014200);
        assertTrue(transaction.getTime().after(new Date(118, 4, 20, 0, 0, 0)));
    }

    @Test(expected = NegativeBalanceError.class)
    public void processTransactionBalanceMustBePositive() throws NegativeBalanceError {
        Account account = new DataStore().loadAccountData(Bank.HANA, "123456789012345");
        Transaction transaction = new Transaction(TransactionType.Withdraw);

        transaction.setAccount(account);
        transaction.setAmount(-99999999);
        try {
            transaction.processTransaction();
        } catch (DataStoreError ex) {
            fail(ex.getClass().getSimpleName());
        }
    }

    @After
    public void restoreFile() throws DataStoreError {
        Account account = dataStore.loadAccountData(Bank.HANA, "123456789012345t");
        dataStore.saveAccountData(account);
    }
}
```

8. User Test

```
public class UserTest {
    private DataStore dataStore;

    @Before
    public void init() {
        dataStore = new DataStore();
    }
    @Test
    public void loadUserTest() {
        User us = dataStore.loadUserData("9512221234567t");
        //No Korean.
        assertEquals(us.getUserName(), "DoHyeon");
        assertEquals(us.getUserId(), "9512221234567");
    }

    @Test
    public void removeCardRemoveCardFromList() {
        User us = dataStore.loadUserData("9512221234567t");
            assertEquals(us.getCardList().length, 2);

        try {
            us.removeCard("1234567890");
        } catch (DataStoreError e) {
            fail("throw DataStoreError: " + e.getMessage());
        }

        for(int i = 0 ; i < us.getCardList().length ; i++){
            assertNotEquals("1234567890", us.getCardList()[i]);
        }
    }
}
```

Activity 2061. Unit Testing

Test Summary

35	0	0	3.886s
tests	failures	ignored	duration

100% successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.swad.cppatm	35	0	0	3.886s	100%

Packages **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
com.swad.cppatm.ATMSystemTest	15	0	0	1.907s	100%
com.swad.cppatm.AccountTest	8	0	0	0.269s	100%
com.swad.cppatm.AdminTest	1	0	0	0.002s	100%
com.swad.cppatm.DataStoreTest	3	0	0	0.008s	100%
com.swad.cppatm.LotteryTest	1	0	0	1.672s	100%
com.swad.cppatm.SystemBalanceTest	2	0	0	0.001s	100%
com.swad.cppatm.TransactionTest	3	0	0	0.023s	100%
com.swad.cppatm.UserTest	2	0	0	0.004s	100%

Class com.swad.cppatm.ATMSystemTest

all > com.swad.cppatm > ATMSystemTest

15	0	0	1.907s
tests	failures	ignored	duration

100% successful

Tests | Standard output | Standard error

Test	Duration	Result
calcBillAccountCorrectlyWorking	0.007s	passed
enterAccountInfoChangesProperty	0.012s	passed
enterAccountInfoRaisesExceptionIfAccountDoesNotExistOnDataStore	0.007s	passed
enterAdminInfoCorrectlyWorking	0.013s	passed
enterBillAmountToWithdrawAsDollarCorrectlyWorking	1.092s	passed
enterBillAsDollarRaisesExceptionIfLengthIsInvalid	0.006s	passed
enterBillRaisesExceptionIfLengthIsInvalid	0.007s	passed
enterLotteryFailedWhenLose	0.433s	passed
enterLotteryMakesTransaction	0.270s	passed
enterNumberOfUsersInputIsZero	0.006s	passed
enterNumberOfUsersSuccessfullyDivideCashAmount	0.007s	passed
enterPasswordFreezesAccountWhenInvalidInputIsRepeatForFiveTimes	0.022s	passed
enterPasswordSuccess	0.007s	passed
enterPeriodToQueryDoesGetTransactions	0.012s	passed
shouldGetCardListCorrectly	0.006s	passed

Class com.swad.cppatm.AccountTest

all > com.swad.cppatm > AccountTest

8	0	0	0.269s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
checkPasswordTest	0.009s	passed
shouldAddTransactionCorrectly	0.008s	passed
shouldChangeBalanceCorrectly	0.018s	passed
shouldFreezeAccountState	0.182s	passed
shouldGetCorrectAccount	0.009s	passed
shouldGetCorrectBalance	0.008s	passed
shouldGetCorrectTransactions	0.009s	passed
shouldSaveAccountCorrectly	0.026s	passed

Class com.swad.cppatm.AdminTest

all > com.swad.cppatm > AdminTest

1	0	0	0.002s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
shouldCheckAdminAccount	0.002s	passed

Class com.swad.cppatm.DataStoreTest

all > com.swad.cppatm > DataStoreTest

3	0	0	0.008s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
loadAccountDataReturnNullWhenAccountIsInvalid	0.001s	passed
loadUserDataReturnNullWhenUserIsInvalid	0.001s	passed
saveAccountDataCreatesFile	0.006s	passed

Class com.swad.cppatm.LotteryTest

all > com.swad.cppatm > LotteryTest

1	0	0	1.672s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
checkResultTest	1.672s	passed

Class com.swad.cppatm.SystemBalanceTest

all > com.swad.cppatm > SystemBalanceTest

2	0	0	0.001s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
throwAdminAlarmExceptionTest	0.001s	passed
throwOverflowBillExceptionTest	0s	passed

Class com.swad.cppatm.TransactionTest

all > com.swad.cppatm > TransactionTest

3	0	0	0.023s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
calcFeeTest	0.005s	passed
processTransactionBalanceMustBePositive	0.006s	passed
processTransactionTest	0.012s	passed

Class com.swad.cppatm.UserTest

all > com.swad.cppatm > UserTest

2	0	0	0.004s
tests	failures	ignored	duration

100% successful

Tests Standard error

Test	Duration	Result
loadUserTest	0s	passed
removeCardRemoveCardFromList	0.004s	passed

Activity 2063. System Testing

Test No.	Name	Description	UseCase	Ref. No.	Pass
1-1	계좌 정보 입력 TEST	입금 버튼을 누르고 은행을 선택한 후 통장 번호를 입력한 경우, 시스템이 해당 계좌를 정상적으로 불러오고 다음 화면으로 넘어가는지 확인한다. 또는 유효하지 않은 통장 번호인 경우 에러 메시지를 띄우며 다음 화면으로 넘어가지 않는지 확인한다.	Deposit	R1.1	P
1-2	잔액 증가 TEST	지폐를 입력하고 입력 버튼을 누른 경우, 거래 결과 화면에 출력되는 잔액에 입금액만큼 잔액이 증가하였는지 확인한다.	Deposit	R1.1	P
1-3	환전 계산 TEST	지폐를 입력하고 입력 버튼을 누른 경우, 거래 결과 화면에 출력되는 잔액에 입금액을 환전한 금액만큼 잔액이 증가하였는지 확인한다.	Deposit Foreign Cash	R1.2	P
2-1	비밀번호 입력 TEST	비밀번호를 입력하고 입력 버튼을 누른 경우, 정상적으로 다음 화면으로 넘어가는지 확인한다. 또는 잘못된 비밀번호를 입력한 경우 에러 메시지를 띄우며 다음 화면으로 넘어가지 않는지 확인한다.	Withdraw	R2.1	P
2-2	잔액 감소 TEST	금액을 입력하고 입력 버튼을 누른 경우, 거래 결과 화면에 출력되는 잔액에 출금액과 수수료의 합만큼 잔액이 감소하였는지 확인한다.	Withdraw, Calculate Fee	R2.1, R5	P
2-3	잔액 부족 에러 TEST	입력한 출금액과 수수료의 합이 출금하고자 하는 계좌의 잔액보다 큰 경우, 화면에 에러 메시지를 띄우며 다음 화면으로 넘어가지 않는지 확인한다.	Withdraw, Calculate Fee	R2.1, R5	P
3-1	잔액 변경 TEST	이체 거래를 정상적으로 진행한 경우, 거래 결과 화면에 출력되는 잔액이 정상적으로 변경되었는지 확인한다. 이체 받은 계좌의 잔액 또한 정상적으로 변경되었는지 확인한다.	Transfer Balance	R3.1	P

3-2	이체액 계산 TEST	총 금액과 인원수를 입력한 경우, 1인당 이체 금액을 정확히 계산해서 화면에 출력하는지 확인한다.	Split Pay	R3.2	P
4-1	거래 내역 출력 TEST	조회 기간을 입력한 경우, 화면에 거래 내역이 출력되는지 확인한다. 잘못된 형식이나 기간을 입력한 경우, 화면에 에러 메시지를 띄우며 다음 화면으로 넘어가지 않는지 확인한다.	Query Transaction List	R4.1	P
4-2	잔액 출력 TEST	잔액 조회 버튼을 누르고 은행과 계좌번호, 올바른 비밀번호를 입력한 경우, 화면에 잔액이 출력되는지 확인한다.	Query Balance	R4.2	P
6-1	카드 목록 조회 TEST	개인식별번호를 입력하고 확인 버튼을 누른 경우, 화면에 해당 유저가 보유한 카드의 목록이 출력되는지 확인한다.	Report Lost Card	R6	P
6-2	카드 정지 TEST	정지할 카드를 선택한 경우, 화면에 카드가 정지되었다는 메세지가 정상적으로 출력되며 유저가 보유한 카드 목록에서 해당 카드가 삭제되는지 확인한다.	Report Lost Card	R6	P
6-3	재발급 신청 TEST	재발급 선택 화면에서 재발급을 재발급을 선택한 경우, 화면에 재발급 신청 메세지가 출력되는지 확인한다.	Report Lost Card	R6	P
6-4	개인식별번호 에러 TEST	잘못된 개인식별번호를 입력할 경우, 화면에 에러 메세지를 띄우며 다음 화면으로 넘어가지 않는지 확인한다.	Report Lost Card	R6	P
7-1	당첨 여부 판단 TEST	주차와 번호를 입력할 경우, 당첨의 여부를 판단해서 당첨이라면 다음 화면으로 넘어가는지 확인한다.	Get Lottery Prize	R7	P
7-2	당첨금 입금 TEST	당첨금 입금 거래를 정상적으로 진행한 경우, 거래 결과 화면에 출력되는 잔액에 당첨금만큼 잔액이 증가하였는지 확인한다.	Get Lottery Prize	R7	P
8-1	거래 제한 TEST	특정 거래에서 비밀번호를 5번 이상 잘못 입력한 경우, 거래가 제한되는지 여부를 확인한다.	Freeze Account	R8	P
8-2	거래 제한	거래 제한이 걸린 계좌로 특정 거래를 시도할 경우, 에러 메시지를 띄우며 다	Freeze	R8	P

	계좌 TEST	음 화면으로 넘어가지 않는지 확인한다.	Account		
9-1	언어 변경 TEST	언어 변경 버튼을 누르고 사용할 언어를 선택한 경우, 초기 화면에 변경된 언어로 인터페이스가 출력되는지 확인한다.	Change Locale	R9	P
10-1	관리자 계정 추가 TEST	관리자를 추가하여 ID를 받은 후, 해당 ID와 비밀번호로 로그인 할 수 있는지 확인한다.	Add Admin	R10	P
10-2	식별번호 생성 TEST	관리자를 여러 번 추가한 후, 각 관리자의 ID가 모두 다른지 확인한다.	Add Admin	R10	P
10-3	관리자 삭제 TEST	관리자 삭제를 선택하고 로그인 화면에서 해당 관리자 ID와 PW를 입력하고 확인을 누르면, 로그인 되지 않음을 확인한다.	Remove Admin	R10	P
10-4	관리자 삭제 에러 TEST	현재 관리자의 수가 한명인 상태에서 관리자 삭제를 선택한 경우, 에러 메시지를 띄우며 삭제되지 않는지 확인한다.	Remove Admin	R10	P
11-1	사용자 기능 제한 TEST	사용자 기능이 활성화된 상태(Active)에서 ATM 상태 변경 버튼을 누른 경우, 사용자 기능이 제한(Frozen)되는지 확인한다.	Toggle ATM State	R11	P
11-2	사용자 기능 제한 해제 TEST	사용자 기능 제한(Frozen) 상태에서 ATM 상태 변경 버튼을 누른 경우, 사용자 기능이 활성화(Active) 되는지 확인한다.	Toggle ATM State	R11	P
11-3	기능 제한 상태 TEST	사용자 기능 제한 상태라면, 어떤 기능도 수행할 수 없는지 확인한다.	Toggle ATM State	R11	P
12-1	지폐 보유량 변경 TEST	지폐 보유량을 입력한 경우, 지폐 보유량 조회를 통해 지폐 보유량이 정상적으로 변경되었는지 확인한다.	Change ATM Balance	R12.1	P

12-2	지폐 보유량 조회 TEST	입금 또는 출금 거래 후, 지폐 보유량이 정상적으로 변경되는지 확인한다.	Query ATM Balance	R12.2	P

Activity 2067. Testing Traceability Analysis