SUPER-
SAFE ATM

~ TEAM 4 ~

201611269 신문기
201610401 손하영
201510283 임진웅

INDEX

1. Revised

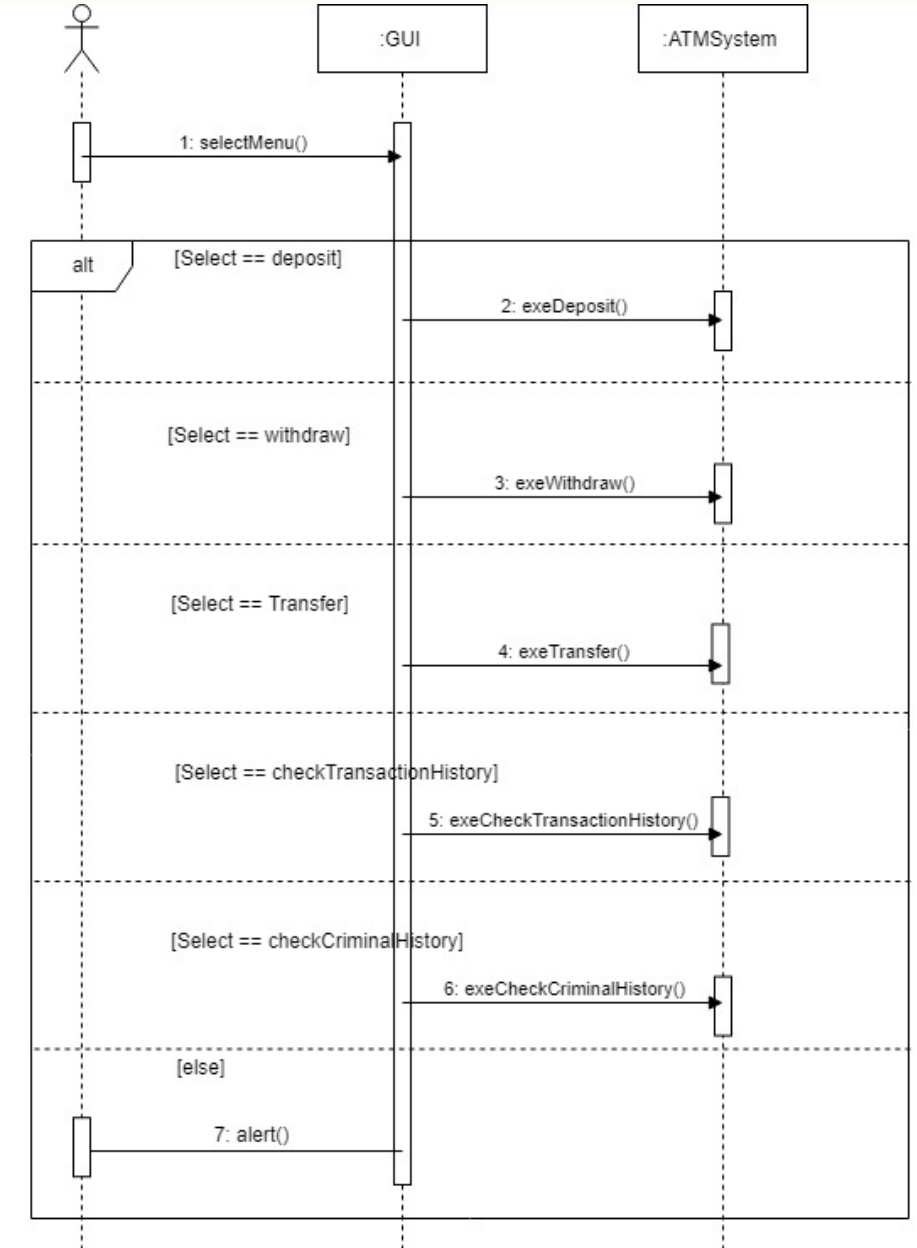
1.1 Functional Requirements

1.2 Interaction Diagram

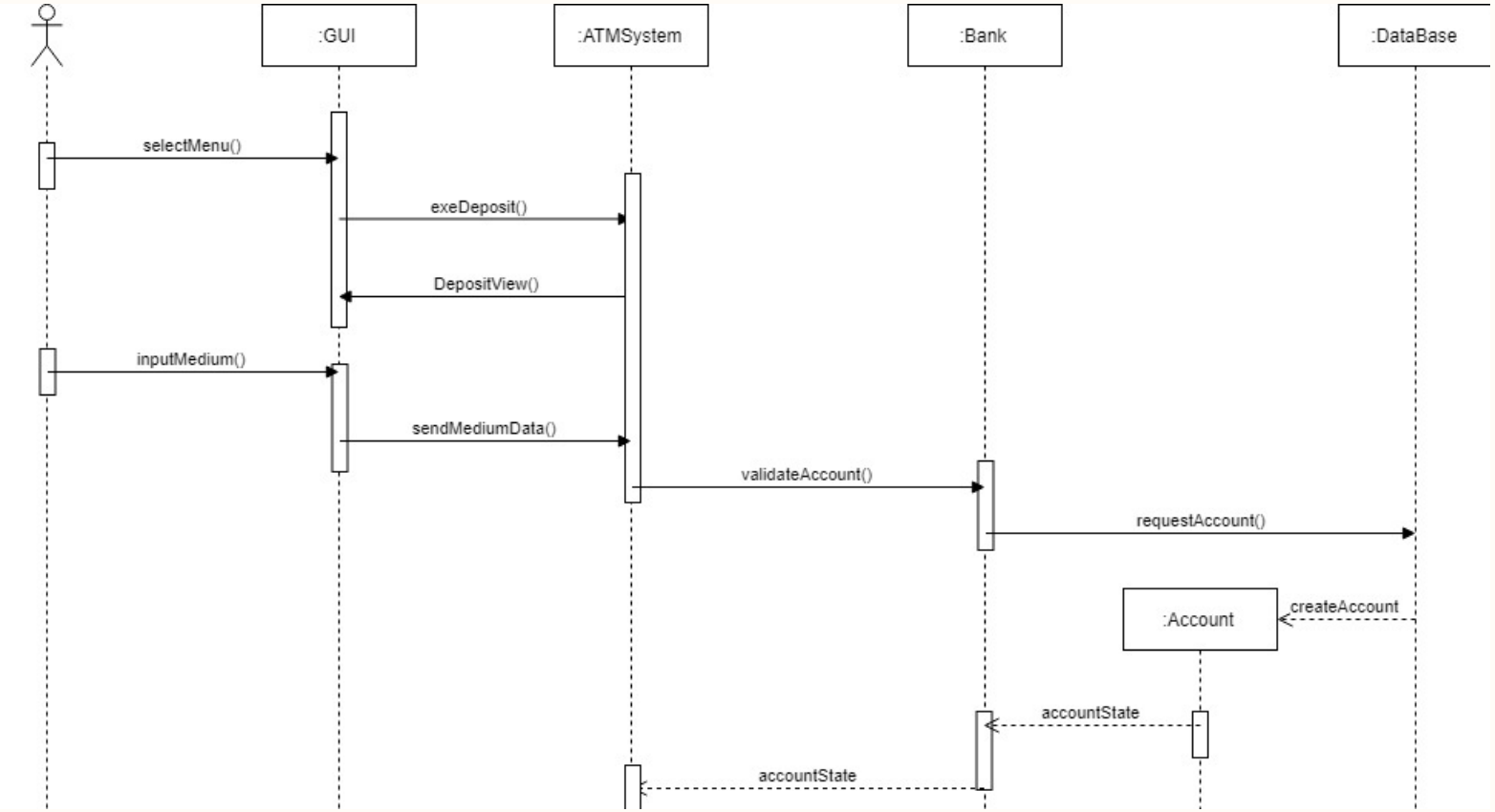
1.1 Functional Requirement

Reference No.	Function	Category
R 1.1	SelectMenu	Evident
R 2.1	inputMedium	Evident
R 3.1	ValidateAccount	Hidden
R 3.2	inputPassword	Evident
R 3.3	checkPassword	Hidden
R 4.1	insertMoney	Evident
R 4.2	insertAmount	Evident
R 4.3	inputAccount	Evident
R 4.4	CalculateBalance	Hidden
R 5.1	exeDeposit	Hidden
R 5.2	exeWithdraw	Hidden
R 5.3	exeTansfer	Hidden
R 5.4	exeCheckTransactionHistory	Hidden
R 5.5	exeCheckCriminalHistory	Hidden
R 6.1	printReceipt	Evident

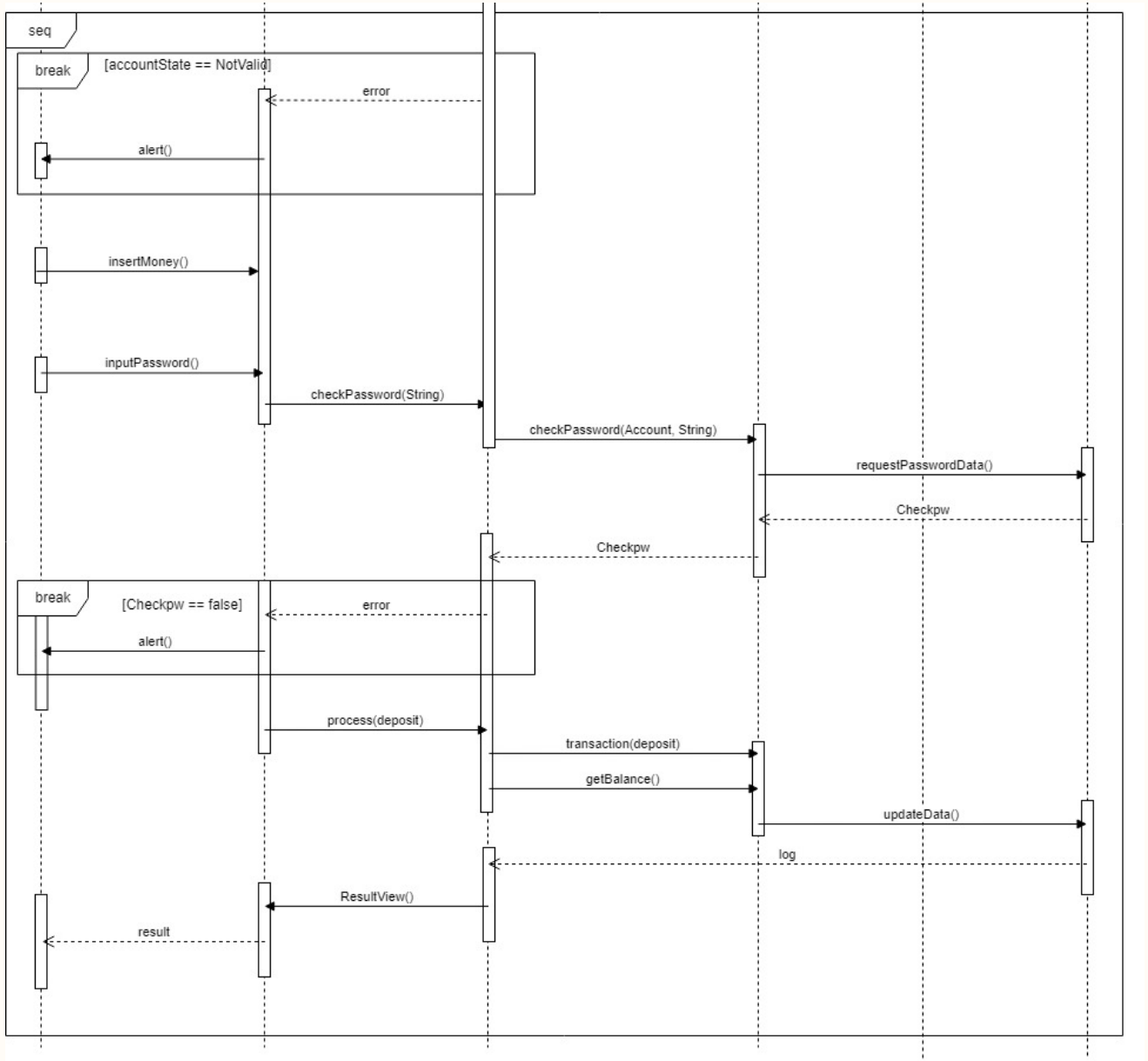
1.2 Interaction Diagram



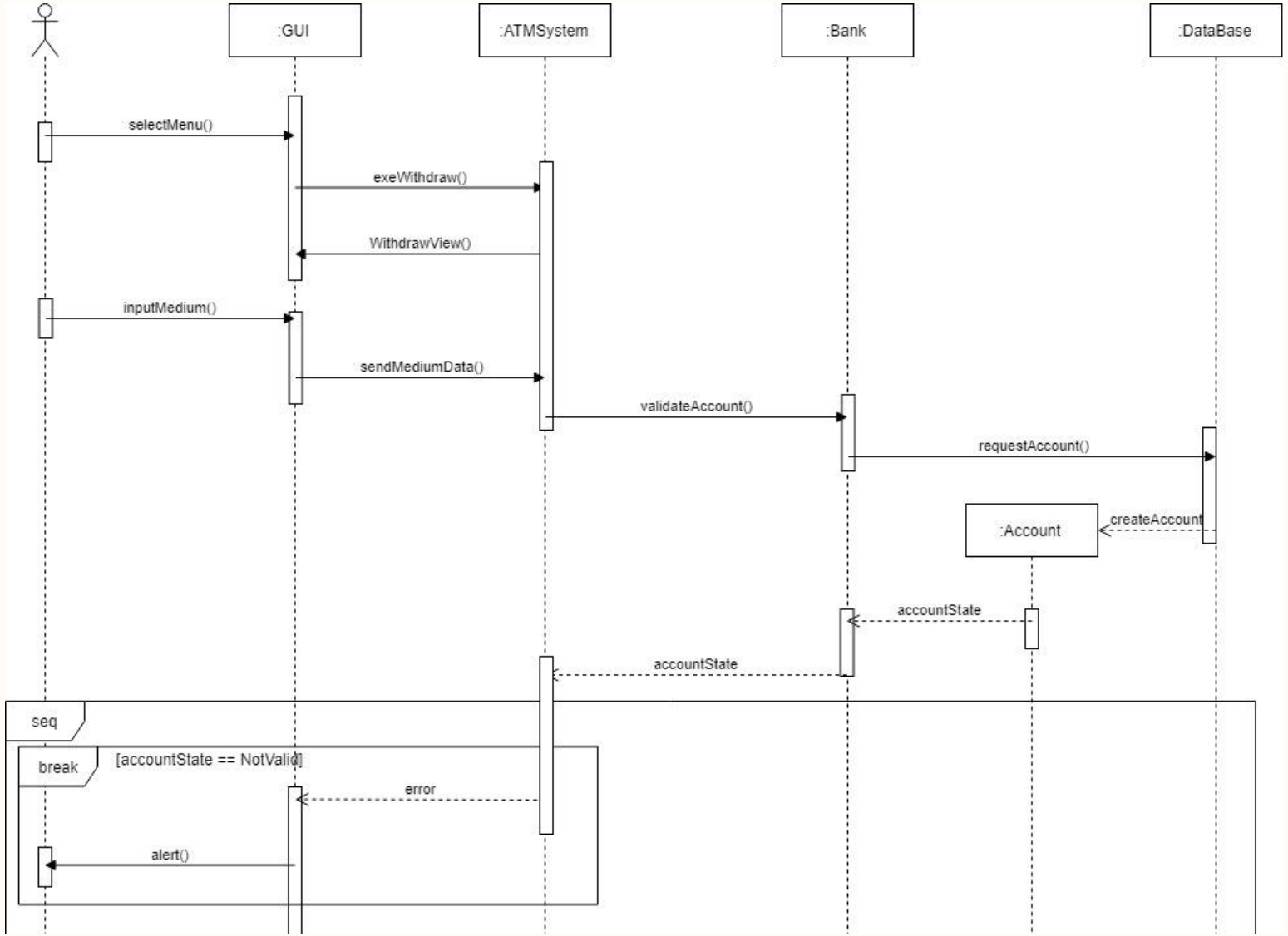
1.2 Interaction Diagram



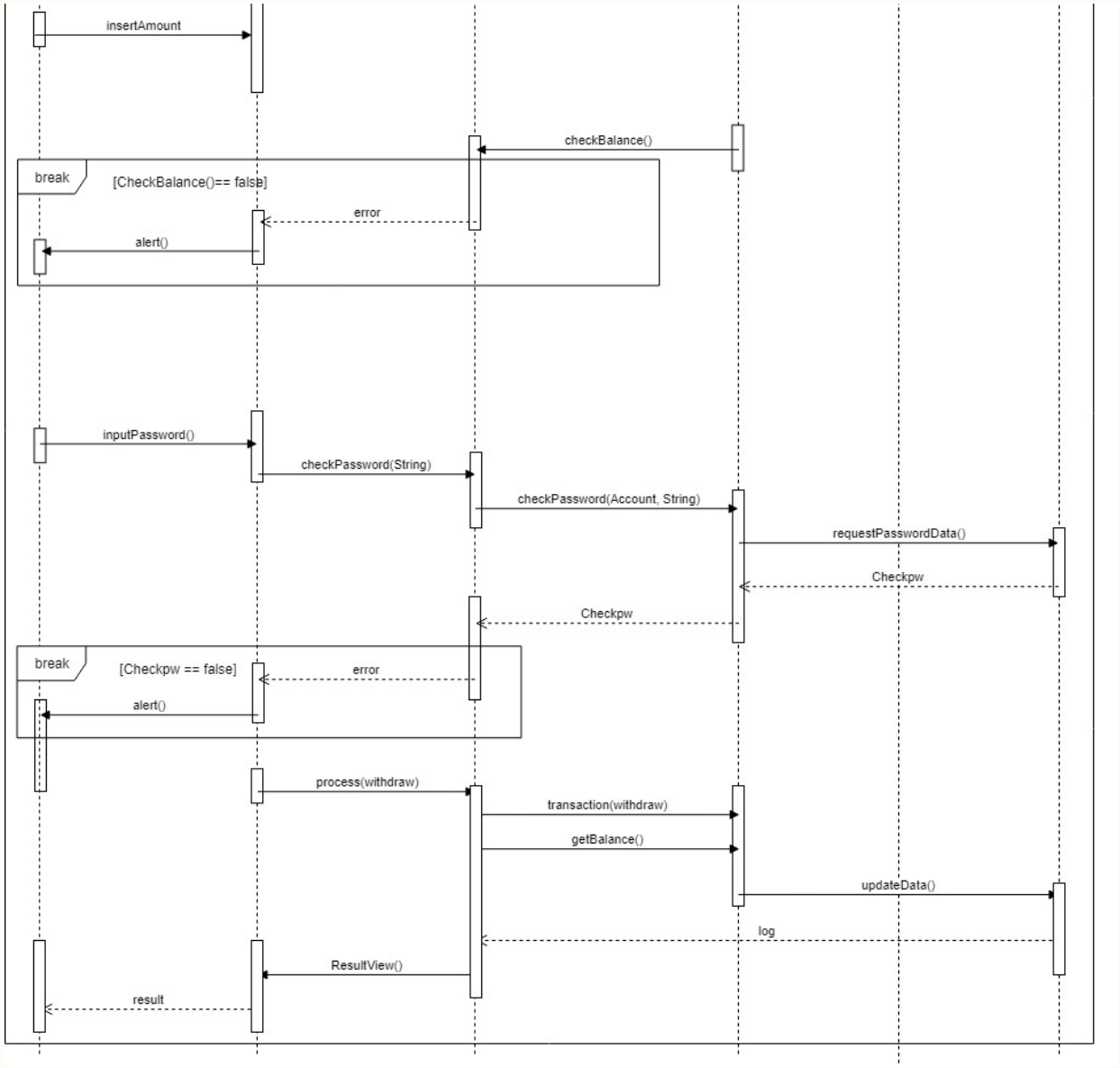
1.2 Interaction Diagram



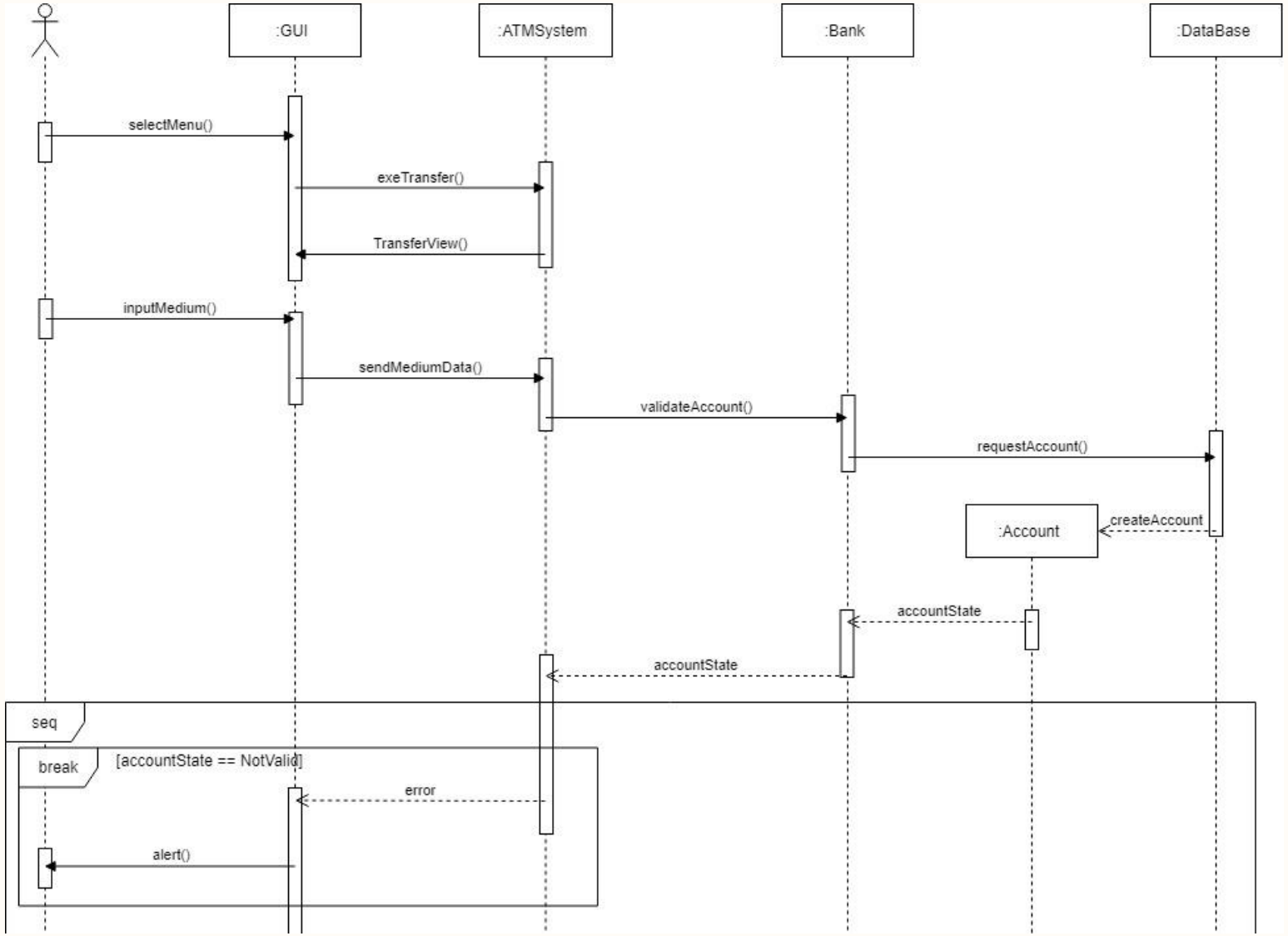
1.2 Interaction Diagram



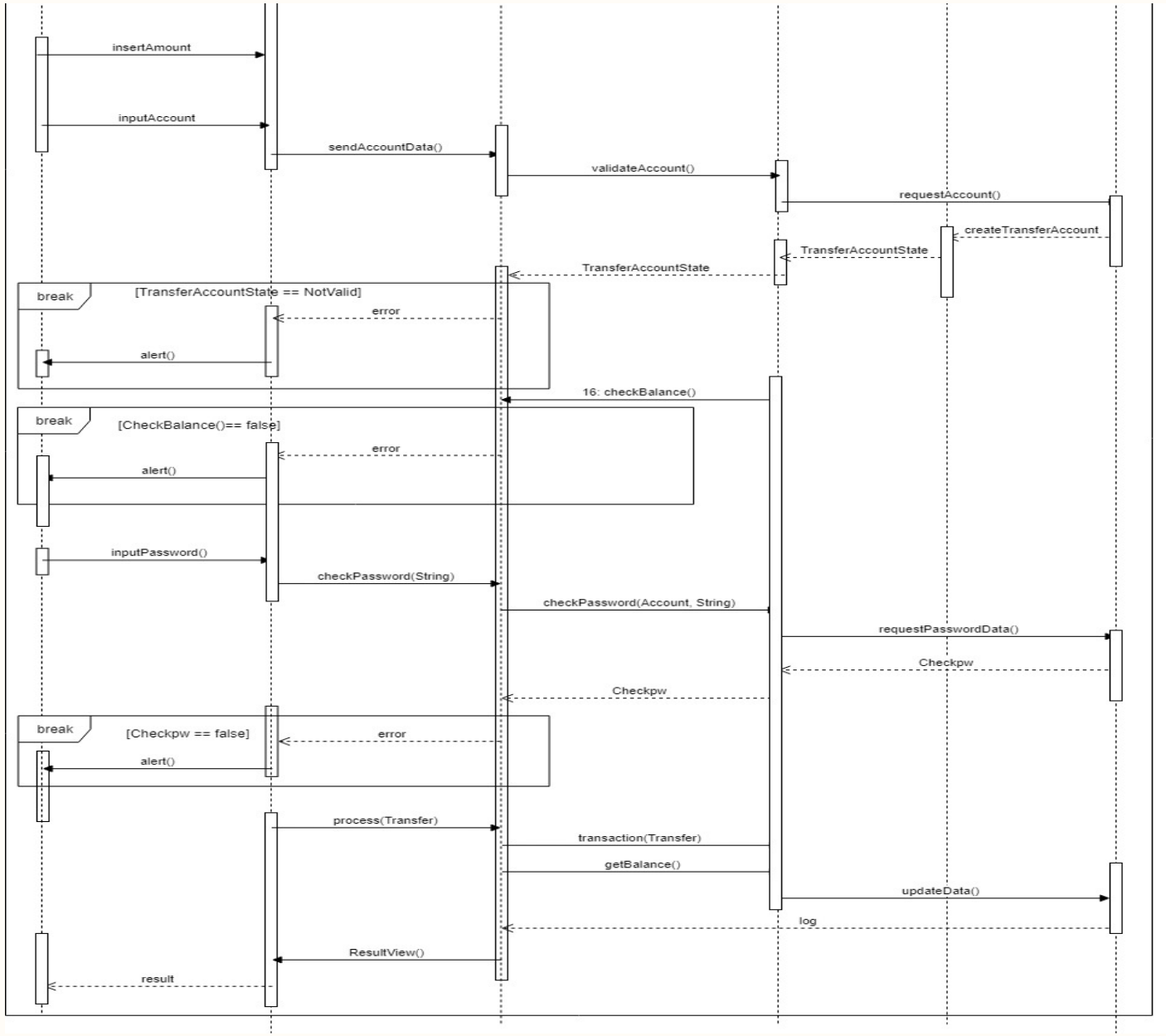
1.2 Interaction Diagram



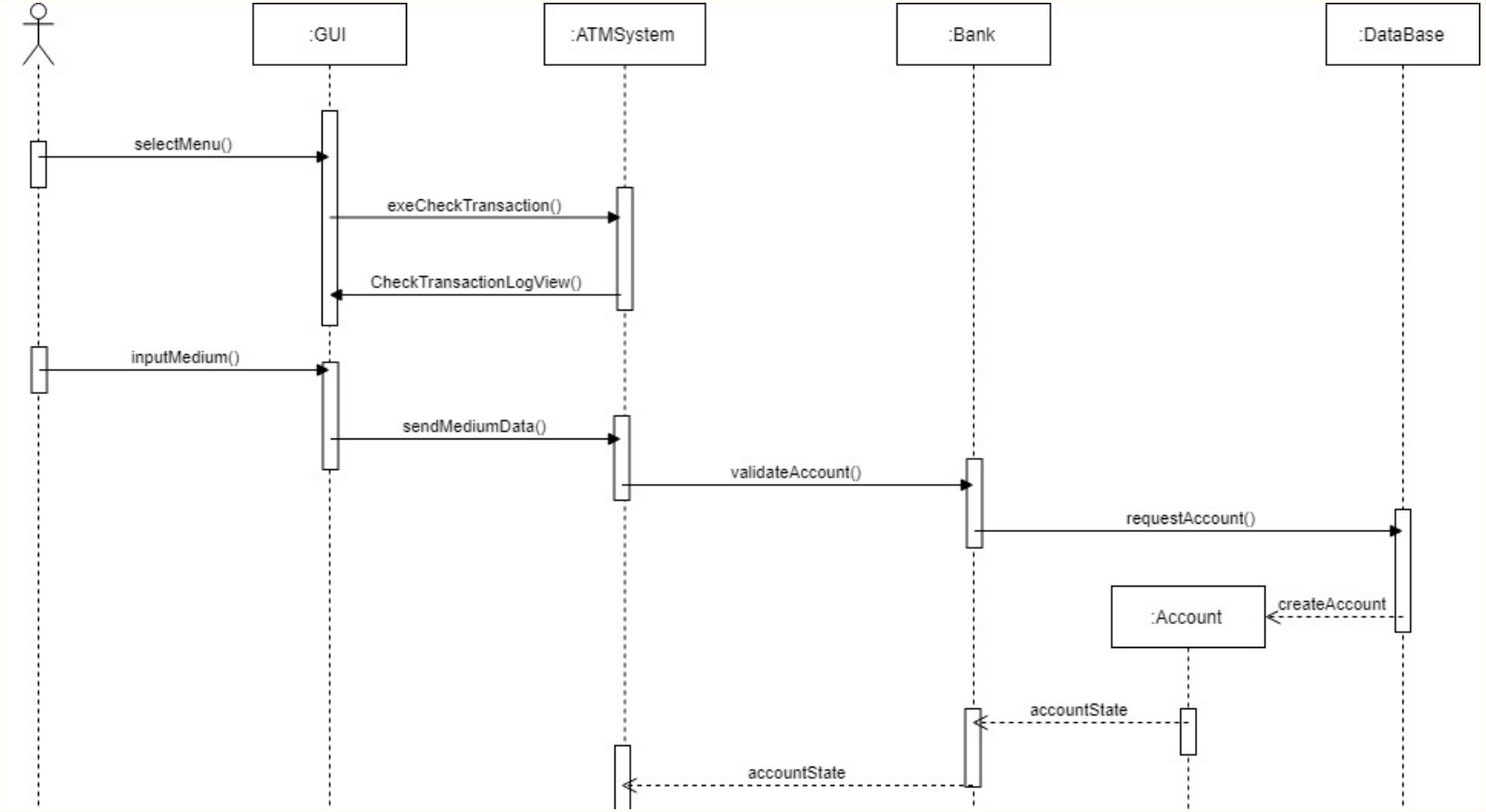
1.2 Interaction Diagram



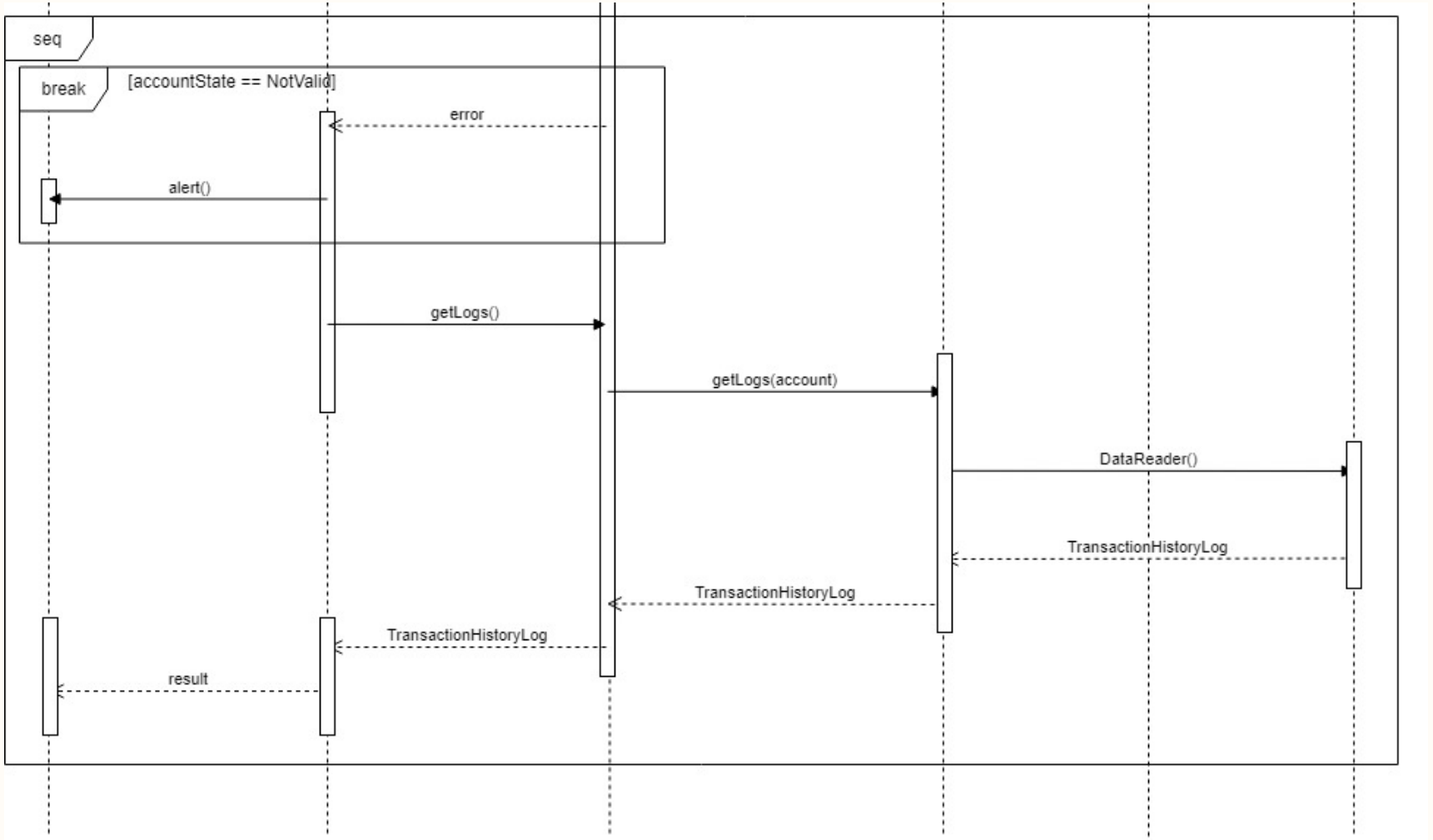
1.2 Interaction Diagram



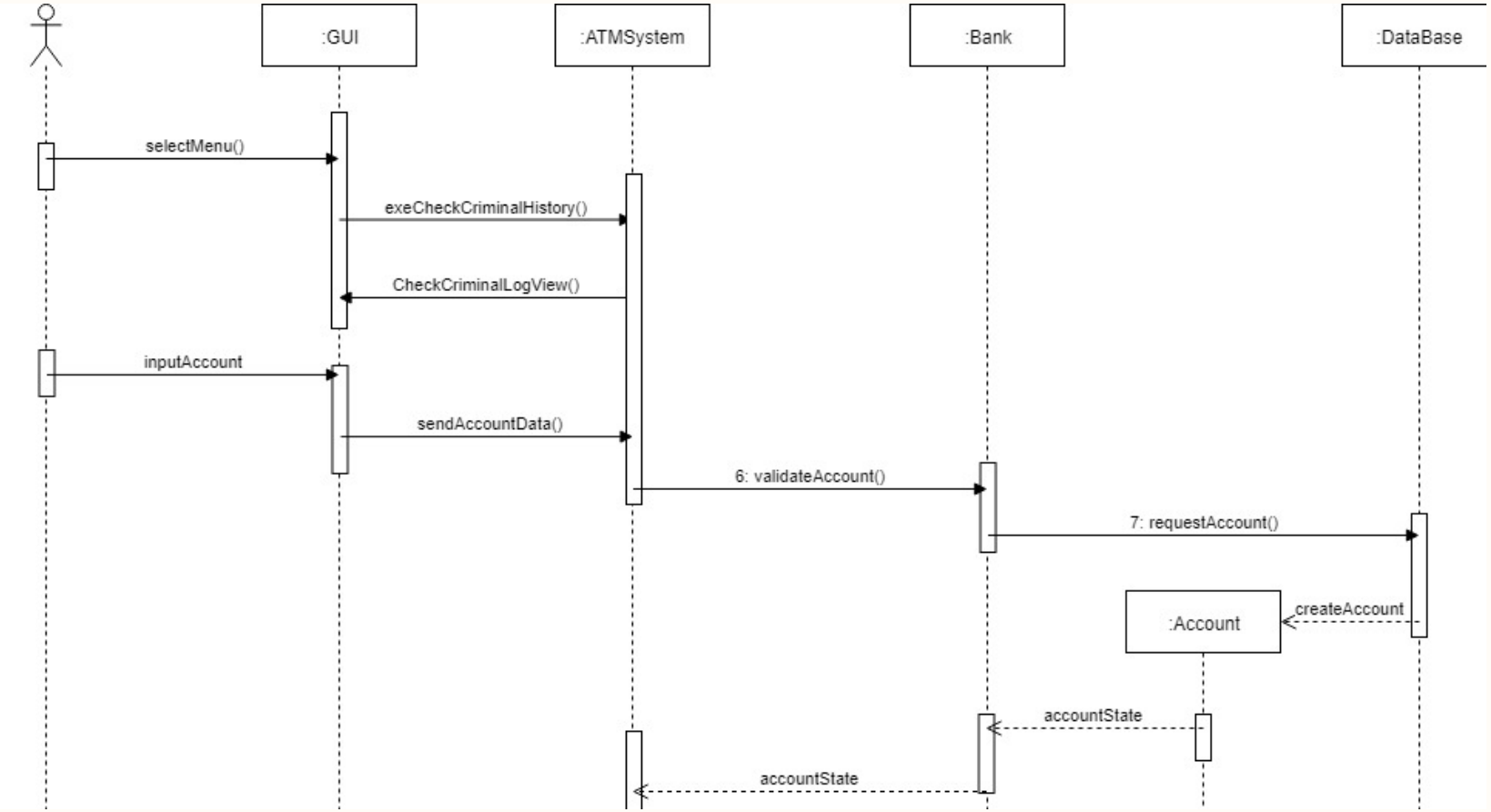
1.2 Interaction Diagram



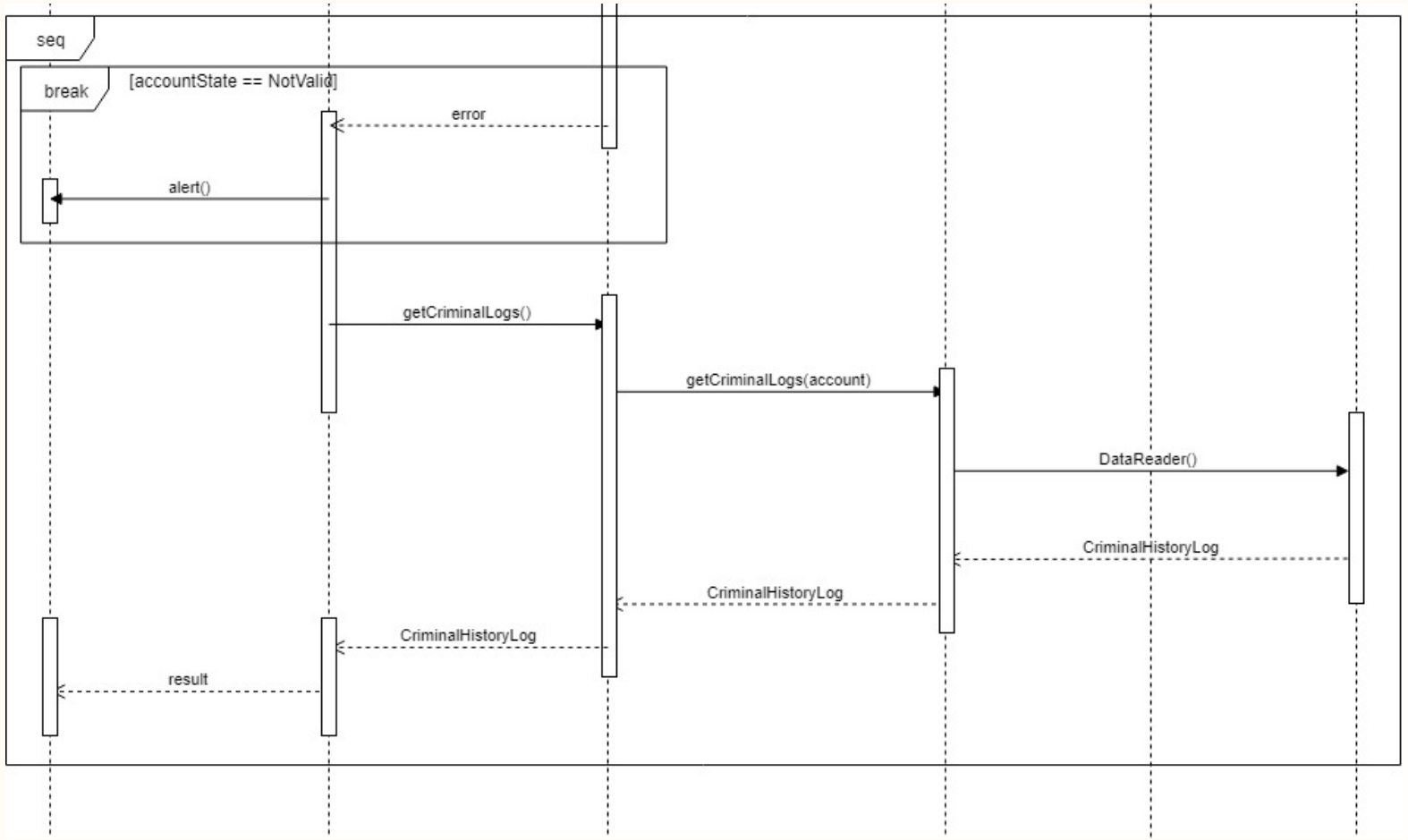
1.2 Interaction Diagram



1.2 Interaction Diagram



1.2 Interaction Diagram



2. Activity 2050-2060

2.1 Act. 2051
2.2 Act. 2052
2.3 Act. 2055
2.4 Act. 2061
2.5 Act. 2063

2.1 Activity 2051 - Class

Type	class
Name	Account
Purpose	계좌 정보를 모아두는 클래스
Overview(class)	Attribute : accountNum, Bank Method : Account(), getBank(), getAccountNum(), getBalance(), checkPassword(), transaction(), getLogs(), getCriminalLogs()를 가지고 있다.
CrossReference	Use Case : R 2.1, R 3.1, R 3.3, R 4.1, R 4.2, R 4.3, R 4.4, R 5.1, R 5.2, R 5.3, R 5.4, R 5.5
Exceptional Courses of Events	N/A

2.1 Activity 2051 - Class

Type	class
Name	AtmSystem
Purpose	ATM 정보를 모아두는 클래스
Overview(class)	Attribute : Bank own, Account src, Account des, amount, fee, banks, transaction Method : AtmSystem(), tansaction(), checkBalance(), checkPassword(), deposit(), withdraw(), transfer(), getLogs(), getCriminalLogs(), getTransaction(), setSrv(), validateSrc(), setDes(), validateDes(), setAmount(), findAccount()
CrossReference	Use Case : R 1.1, R 2.1, R.3.1, R 3.2, R 3.3, R 4.1, R 4.2, R 4.3, R 4.4, R 5.1, R 5.2, R 5.3, R 5.4, R 5.5, R 6.1
Exceptional Courses of Events	N/A

2.1 Activity 2051 - Class

Type	class
Name	Bank
Purpose	Bank 정보를 모아두는 클래스
Overview(class)	Attribute : name, path Method : Bank(), fileReader(), validateAccount(), checkPassword(), getBalance(), transaction(), getLogs(), getCriminalLogs()
CrossReference	Use Case : R 3.1, R 3.3, R 5.1, R 5.2, R 5.3, R 5.4, R 5.5
Exceptional Courses of Events	N/A

2.1 Activity 2051 - Method (class Account)

Type	Method
Name	Account()
Purpose	Account의 객체 생성한다.
CrossReference	Use Case : R 2.1, R 3.1, R 3.3, R 4.1, R 4.2, R 4.3, R 4.4, R 5.1, R 5.2, R 5.3, R 5.4, R 5.5
Input(Method)	String accountNum, Bank bank
Output(Method)	-
Abstract Operation(Method)	Account의 객체를 생성한다.
Exceptional Courses of Events	-

Type	Method
Name	getBank()
Purpose	Account의 소속 bank를 확인한다.
CrossReference	Use Case R 4.4
Input(Method)	-
Output(Method)	Bank bank
Abstract Operation(Method)	account.bank를 반환한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Account)

Type	Method
Name	getAccountNum
Purpose	Account의 계좌번호를 확인한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	String accountNum
Abstract Operation(Method)	account.accountNum을 반환한다.
Exceptional Courses of Events	

Type	Method
Name	getBalance
Purpose	계좌 잔액을 확인한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	int bank.getBalance(account: this)
Abstract Operation(Method)	bank DB에서 balance를 불러와 반환한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Account)

Type	Method
Name	checkPassword()
Purpose	비밀번호가 유효한지 확인한다.
CrossReference	Use Case
Input(Method)	String pw
Output(Method)	boolean bank.checkPassword(account: this, pw)
Abstract Operation(Method)	bankDB에서 비밀번호 일치여부를 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	transaction()
Purpose	거래(입금/출금/송금)
CrossReference	Use Case
Input(Method)	int amount, Sting msg
Output(Method)	bank.transaction(account: this, amount, msg)
Abstract Operation(Method)	이 계좌에서 amount만큼 거래하고 log에 msg를 저장한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Account)

Type	Method
Name	getLogs()
Purpose	거래내역을 조회한다.
CrossReference	Use Case :
Input(Method)	-
Output(Method)	ArrayList<String> bank.getLogs(account: this)
Abstract Operation(Method)	해당 계좌의 log를 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	getCriminalLogs()
Purpose	범죄이력을 조회한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	ArrayList<String> bank.getCriminalLogs(account: this)
Abstract Operation(Method)	계좌의 범죄 이력과 횟수를 반환한다,
Exceptional Courses of Events	

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	AtmSystem()
Purpose	AtmSystem의 객체를 생성한다.
CrossReference	Use Case : R 1.1, R 2.1, R.3.1, R 3.2, R 3.3, R 4.1, R 4.2, R 4.3, R 4.4, R 5.1, R 5.2, R 5.3, R 5.4, R 5.5, R 6.1
Input(Method)	Bank own
Output(Method)	-
Abstract Operation(Method)	AtmSystem의 객체를 생성한다.
Exceptional Courses of Events	-

Type	Method
Name	transaction()
Purpose	이용할 서비스를 선택한다.
CrossReference	Use Case R 1.1
Input(Method)	-
Output(Method)	-
Abstract Operation(Method)	button 입력에 따라 선택한 서비스를 실행한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	checkBalance()
Purpose	계좌 잔고가 출금 또는 송금할 금액보다 많은지 확인한다.
CrossReference	Use Case : R 5.2 R.5.3
Input(Method)	-
Output(Method)	boolean (amount + fee) < arc.getBalance()
Abstract Operation(Method)	amount + fee와 src.balance를 비교한 결과를 반환한다,
Exceptional Courses of Events	-

Type	Method
Name	checkPassword
Purpose	비밀번호를 확인한다.
CrossReference	Use Case
Input(Method)	String pw
Output(Method)	boolean src.checkPassword(pw);
Abstract Operation(Method)	입력한 비밀번호가 src의 비밀번호와 일치하는지 확인한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	deposit()
Purpose	src계좌에 입금한다.
CrossReference	Use Case R5.1
Input(Method)	-
Output(Method)	-
Abstract Operation(Method)	계좌의 balance를 amount - fee만큼 증가시키고 log에 msg와 해당 거래내용을 업데이트한다.
Exceptional Courses of Events	-

Type	Method
Name	withdraw()
Purpose	src에서 출금한다.
CrossReference	Use Case R 5.2
Input(Method)	-
Output(Method)	-
Abstract Operation(Method)	계좌의 balance를 amount + fee만큼 감소시키고 log에 msg와 해당 거래내용을 업데이트한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	transfer()
Purpose	src에서 des로 송금한다.
CrossReference	Use Case 5.3
Input(Method)	-
Output(Method)	-
Abstract Operation(Method)	src 계좌의 balance를 amount + fee만큼 감소시키고 log에 msg와 해당 거래내용을 업데이트한다 des 계좌의 balance를 amount 만큼 증가시키고 log에 msg와 해당 거래내용을 업데이트한다
Exceptional Courses of Events	-

Type	Method
Name	getLogs()
Purpose	계좌 거래내역을 조회한다.
CrossReference	Use Case 5.4
Input(Method)	-
Output(Method)	String[][] contents
Abstract Operation(Method)	해당 계좌의 거래내역 contents를 반환한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	getCriminalLogs()
Purpose	계좌 범죄내역을 조회한다.
CrossReference	Use Case 5.5
Input(Method)	-
Output(Method)	String[][] contents
Abstract Operation(Method)	해당 계좌의 범죄내역 contents를 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	setTransaction()
Purpose	이용할 서비스를 선택한다.
CrossReference	Use Case R 1.1
Input(Method)	String transaction
Output(Method)	-
Abstract Operation(Method)	String transaction을 입력받아 AtmSystem의 transaction을 세팅한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	getTransaction()
Purpose	현재 이용 서비스를 반환한다.
CrossReference	Use Case R 1.1
Input(Method)	-
Output(Method)	String Transaction
Abstract Operation(Method)	현재 transaction을 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	setSrc()
Purpose	이용할 계좌를 세팅한다.
CrossReference	Use Case
Input(Method)	String accountNum
Output(Method)	-
Abstract Operation(Method)	findaccount(accountNum)의 결과를 this.src에 저장한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	validateSrc()
Purpose	src계좌의 유효성을 확인한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	boolean src != null
Abstract Operation(Method)	src계좌의 유효성에 따라 boolean 값을 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	setDes
Purpose	송금 대상 계좌를 세팅한다.
CrossReference	Use Case
Input(Method)	String accountNum
Output(Method)	-
Abstract Operation(Method)	findaccount(accountNum)의 결과를 this.des에 저장한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	validateDes()
Purpose	des계좌의 유효성을 확인한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	boolean des != null
Abstract Operation(Method)	des계좌의 유효성에 따라 boolean 값을 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	setAmount()
Purpose	서비스 이용 금액을 세팅한다.
CrossReference	Use Case
Input(Method)	int amount
Output(Method)	-
Abstract Operation(Method)	this.amount에 amount값을 저장한다.
Exceptional Courses of Events	.-

2.1 Activity 2051 - Method (class ATMSystem)

Type	Method
Name	findAccount()
Purpose	각 bank DB에 account 가 존재하는지 찾는다.
CrossReference	Use Case
Input(Method)	String AccountNum
Output(Method)	Account ret
Abstract Operation(Method)	각 bank DB 에서 accountNum을 검색하고 존재하면 ret을 반환한다.
Exceptional Courses of Events	어느 bank DB에도 존재하지 않는 accountNum의 경우 null을 반환한다.

2.1 Activity 2051 - Method (class Bank)

Type	Method
Name	Bank()
Purpose	Bank의 객체를 생성한다.
CrossReference	Use Case
Input(Method)	String name
Output(Method)	-
Abstract Operation(Method)	Bank의 객체를 생성한다.
Exceptional Courses of Events	-

Type	Method
Name	fileReader()
Purpose	파일로 부터 정보를 읽어온다.
CrossReference	Use Case R 5.4, R 5.5
Input(Method)	String path
Output(Method)	AttrayList<String> lines
Abstract Operation(Method)	파일로 저장된 log를 읽어와 반환한다.
Exceptional Courses of Events	path가 올바르지 않을시 Error

2.1 Activity 2051 - Method (class Bank)

Type	Method
Name	validateAccount()
Purpose	계좌가 유효한지 확인한다.
CrossReference	Use Case
Input(Method)	String accountNum
Output(Method)	Account ret
Abstract Operation(Method)	bank DB에 accountNum이 존재하는지 확인하고 존재하면 해당 계좌를 반환한다.
Exceptional Courses of Events	bank DB에 accountNum이 존재하는지 확인하고 존재하지 않으면 null을 반환한다.

Type	Method
Name	checkPassword()
Purpose	계좌의 비밀번호가 일치하는지 확인한다.
CrossReference	Use Case
Input(Method)	Account account, String password
Output(Method)	boolean pw.equals(password)
Abstract Operation(Method)	account의 비밀번호와 입력한 password가 일치하는지 확인하고 boolean값을 반환한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Bank)

Type	Method
Name	getBalance()
Purpose	계좌의 잔액을 확인한다.
CrossReference	Use Case R 5.1, 5.3
Input(Method)	Account account
Output(Method)	int balance
Abstract Operation(Method)	account의 balance를 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	transaction()
Purpose	거래 서비스를 실행한다.
CrossReference	Use Case R 1.1, R 5.1, R 5.2, R 5.3
Input(Method)	Account account, int amount, String msg
Output(Method)	-
Abstract Operation(Method)	account의 log파일에 거래 시간, 거래량, 거래 유형을 기록한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Bank)

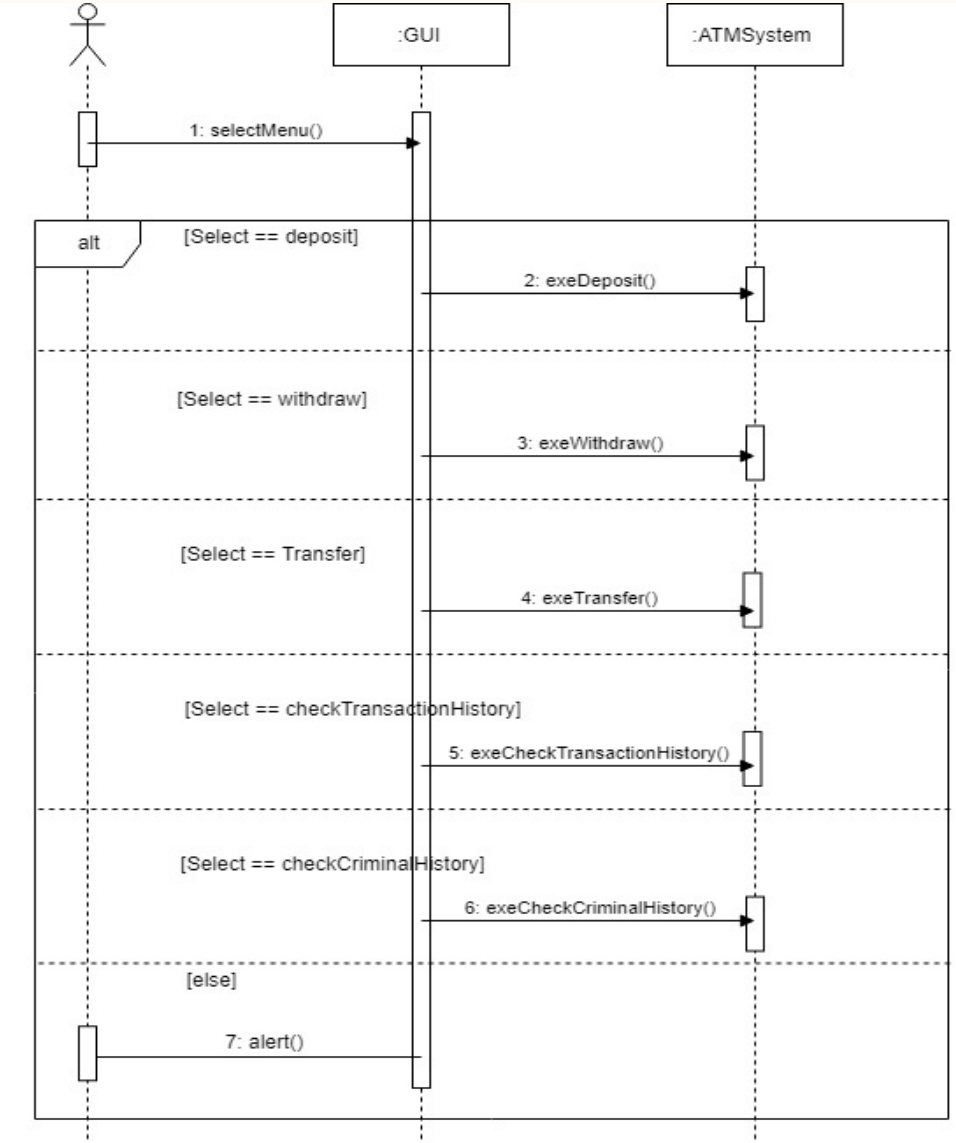
Type	Method
Name	getLogs()
Purpose	거래내역을 출력한다.
CrossReference	Use Case R 5.4
Input(Method)	Account account
Output(Method)	ArrayList<String> this.fileReader(path)
Abstract Operation(Method)	account의 거래내역 로그파일을 읽어와 반환한다.
Exceptional Courses of Events	-

Type	Method
Name	getCriminalLogs()
Purpose	거래내역을 출력한다.
CrossReference	Use Case R 5.5
Input(Method)	Account account
Output(Method)	ArrayList<String> this.fileReader(path)
Abstract Operation(Method)	account의 범죄 이력 로그파일을 읽어와 반환한다.
Exceptional Courses of Events	-

2.1 Activity 2051 - Method (class Bank)

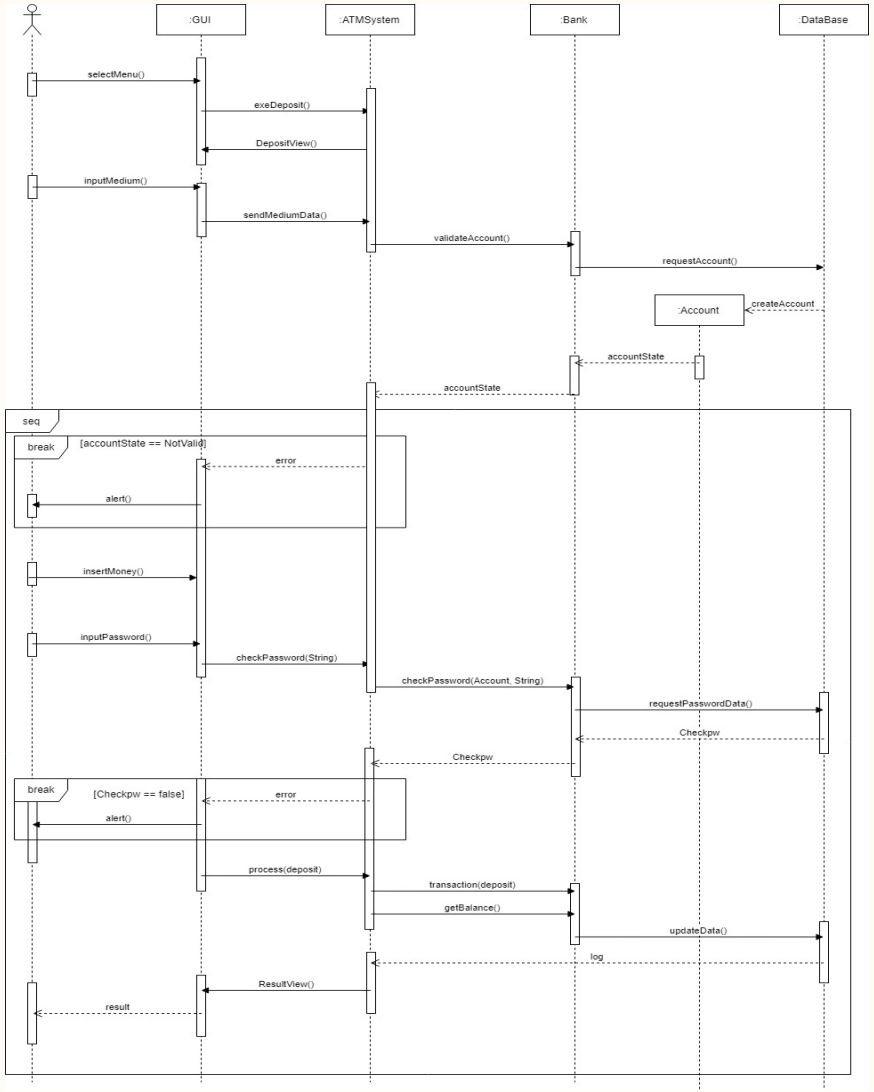
Type	Method
Name	getName()
Purpose	은행의 이름을 확인한다.
CrossReference	Use Case
Input(Method)	-
Output(Method)	-
Abstract Operation(Method)	Bank의 name을 반환한다.
Exceptional Courses of Events	-

2.2 Activity 2052



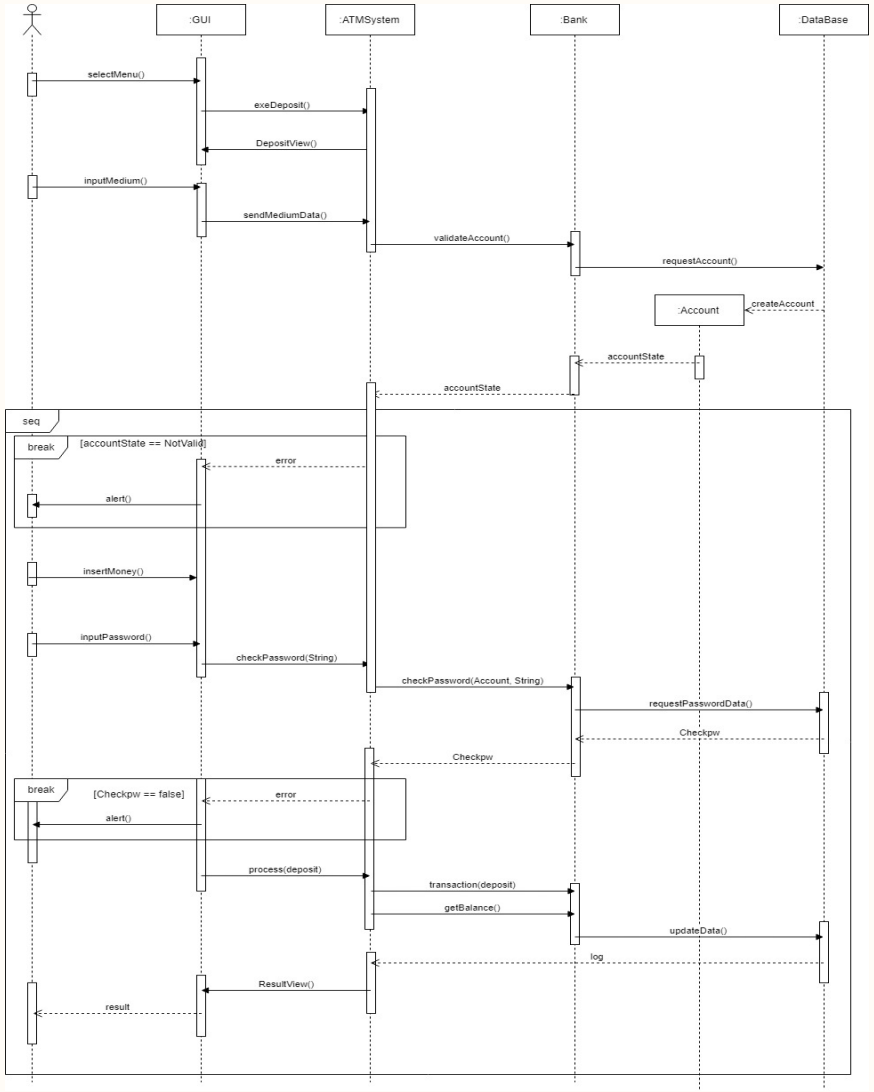
Name	SelectMenu
Type	GUI
Responsibilities	입금, 출금, 송금, 조회, 범죄이력 버튼 중 하나를 누른다.
Cross Reference	R 1.1
Notes	입금, 출금, 송금, 조회, 범죄이력 조회 버튼 중 하나를 누른다.
Post-Condition	선택한 버튼의 기능으로 진행하고 계좌를 입력할 수 있다.
Pre-Condition	ATM이 켜진 상황이어야 한다.

2.2 Activity 2052



Name	DepositView
Type	GUI
Responsibilities	입금을 진행하는 화면이다.
Cross Reference	R 5.1
Notes	입금을 진행하는 화면이다.
Post-Condition	계좌의 비밀번호를 입력할 수 있다.
Pre-Condition	입금 버튼을 누른 상황이어야 한다.

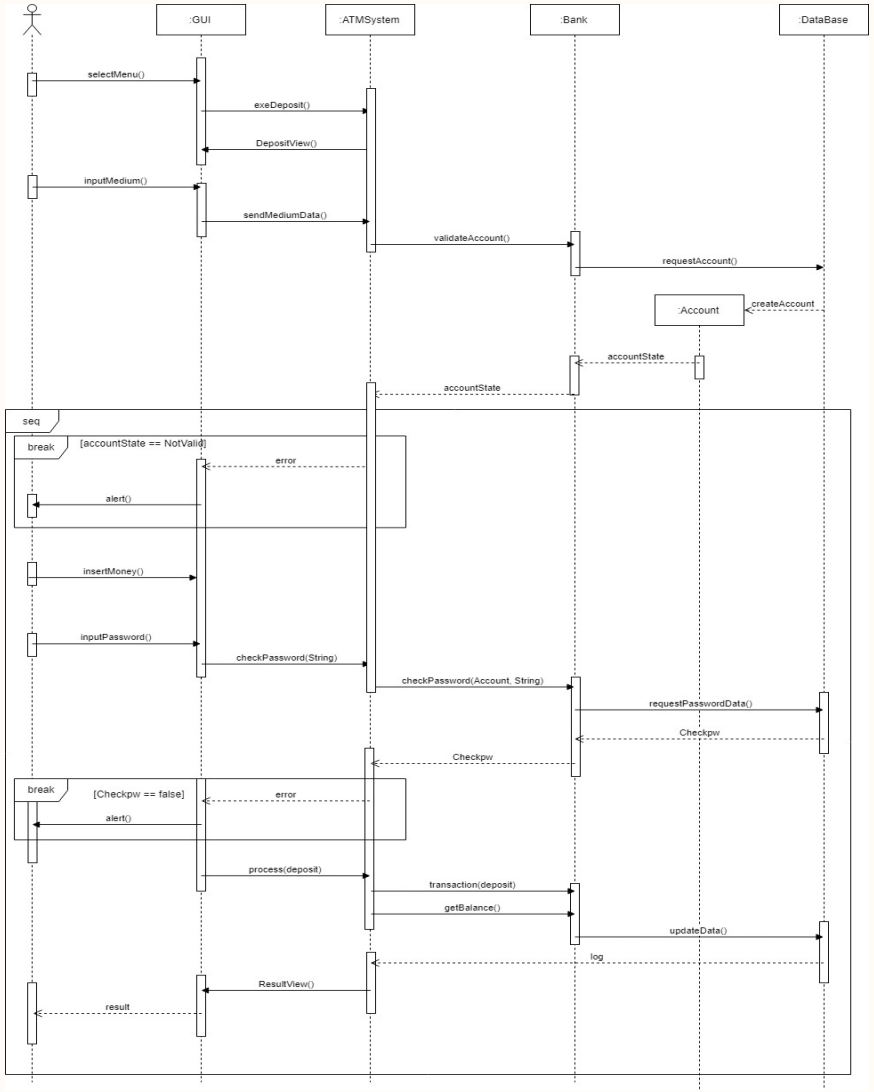
2.2 Activity 2052



Name	inputMedium
Type	GUI
Responsibilities	매체의 계좌를 입력할 수 있다.
Cross Reference	R 2.1
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효할 경우 돈을 입력할 수 있다.
Pre-Condition	입금 버튼을 누른 상황이어야 한다.

Name	alert “Invalid Account”
Type	GUI
Responsibilities	계좌가 유효하지 않음을 알린다.
Cross Reference	R 3.1
Notes	계좌가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 계좌를 입력한다.

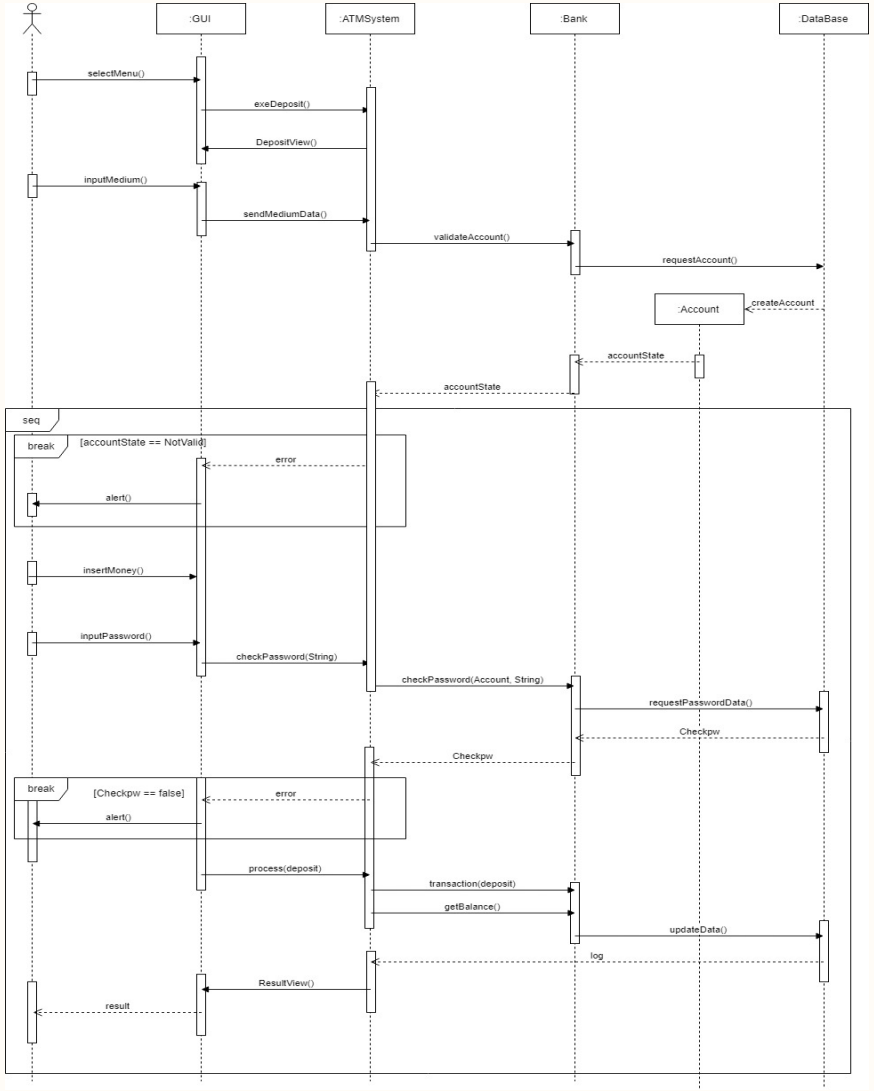
2.2 Activity 2052



Name	inputPassword
Type	GUI
Responsibilities	계좌의 비밀번호를 입력할 수 있다.
Cross Reference	R 3.2
Notes	비밀번호를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	비밀번호가 유효할 경우 입금이 계속 진행되고 결과창이 뜬다.
Pre-Condition	입금할 돈을 입력받은 상황이어야 한다.

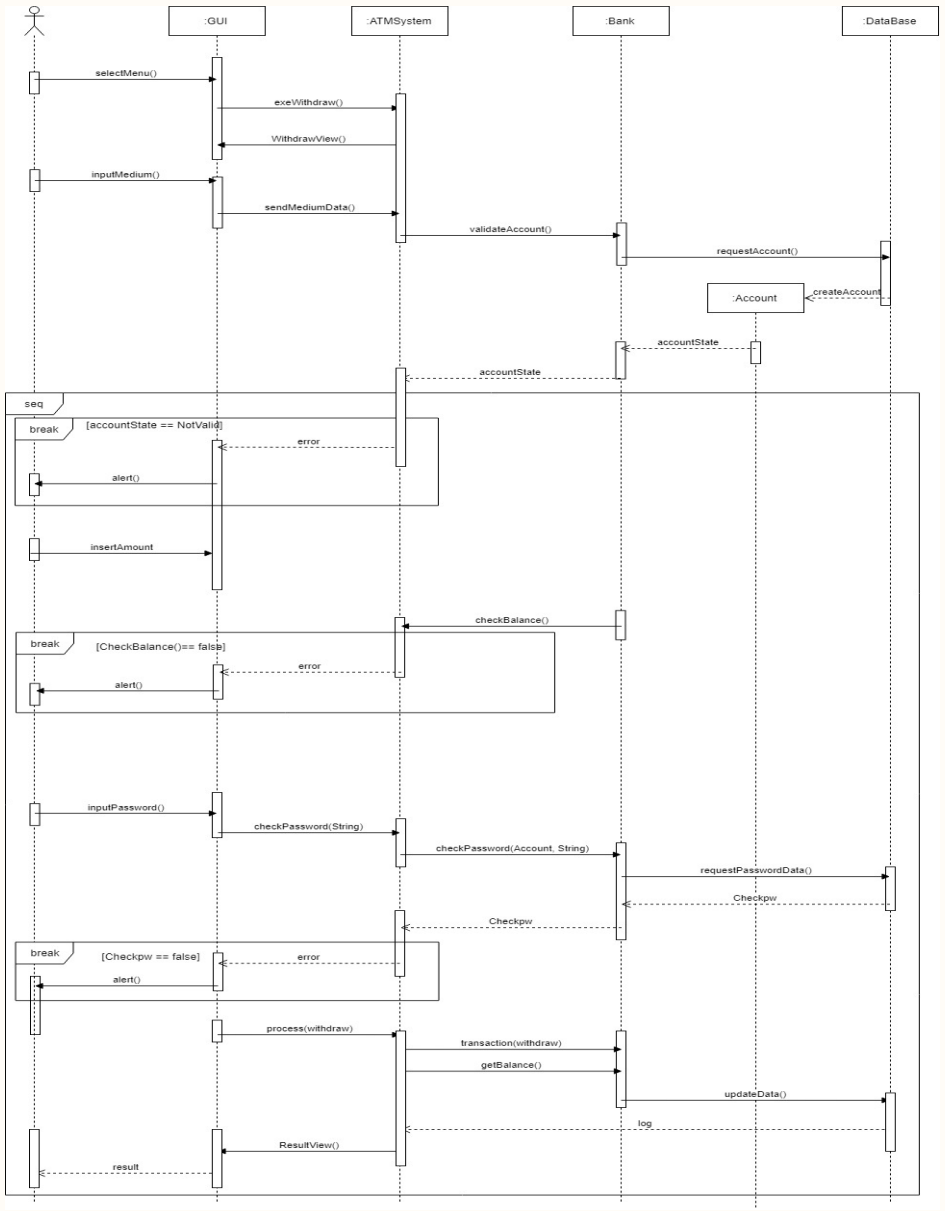
Name	alert “wrong password”
Type	GUI
Responsibilities	비밀번호가 유효하지 않음을 알린다.
Cross Reference	R 3.3
Notes	비밀번호가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 비밀번호를 입력한다.

2.2 Activity 2052



Name	ResultView
Type	GUI
Responsibilities	입금이 완료된 후 결과와 내역을 보여준다.
Cross Reference	R 6.1
Notes	작업, 금액, 잔액의 입금 내역을 보여준다.
Post-Condition	끝내기 버튼을 누를 수 있다.
Pre-Condition	유효한 비밀번호를 입력하고 입금이 계속 진행된다.

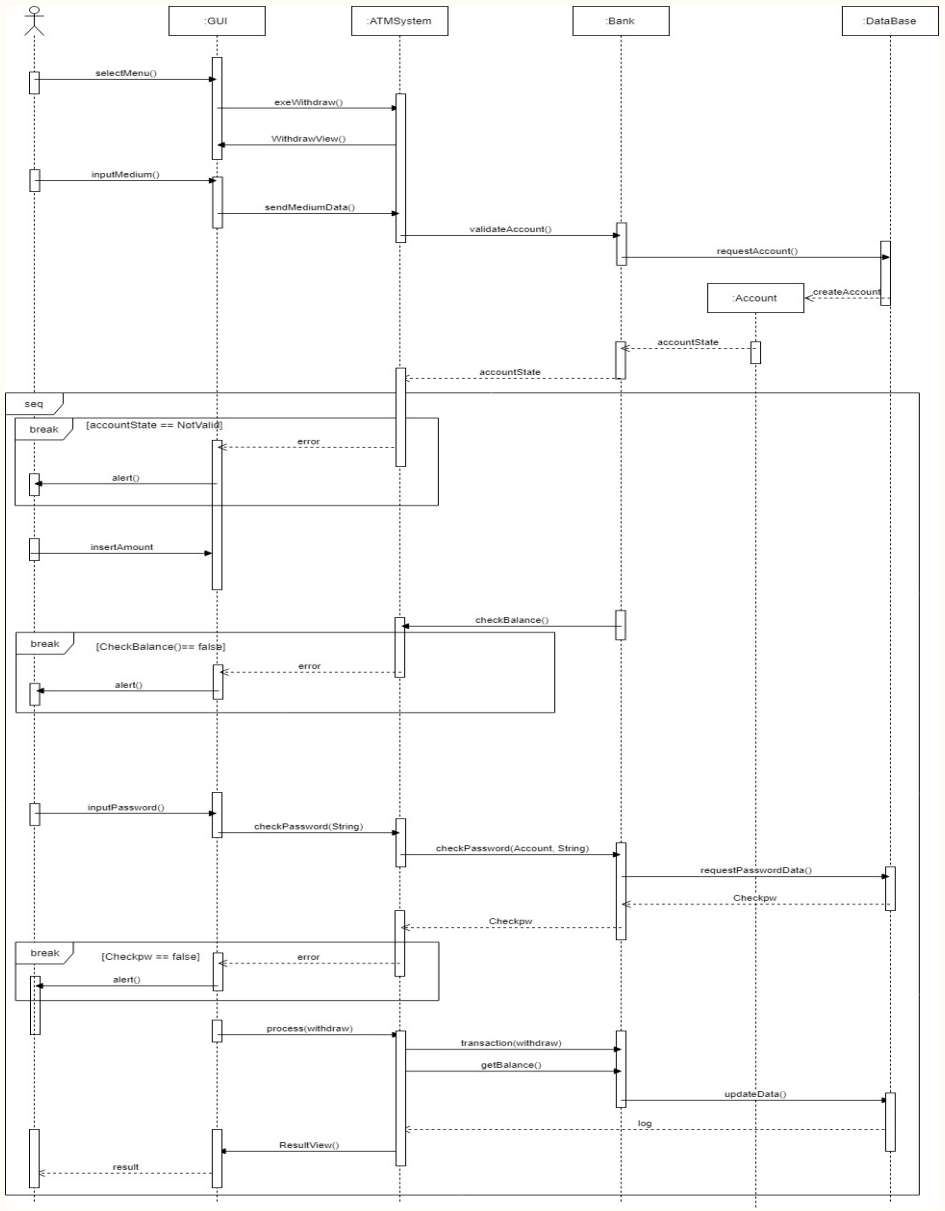
2.2 Activity 2052



Name	WithdrawView
Type	GUI
Responsibilities	출금을 진행하는 화면이다.
Cross Reference	R 5.2
Notes	출금을 진행하는 화면이다.
Post-Condition	계좌의 비밀번호를 입력할 수 있다.
Pre-Condition	출금 버튼을 누른 상황이어야 한다.

Name	inputMedium
Type	GUI
Responsibilities	매체의 계좌를 입력할 수 있다.
Cross Reference	R 2.1
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효할 경우 출금할 돈을 입력할 수 있다.
Pre-Condition	출금 버튼을 누른 상황이어야 한다.

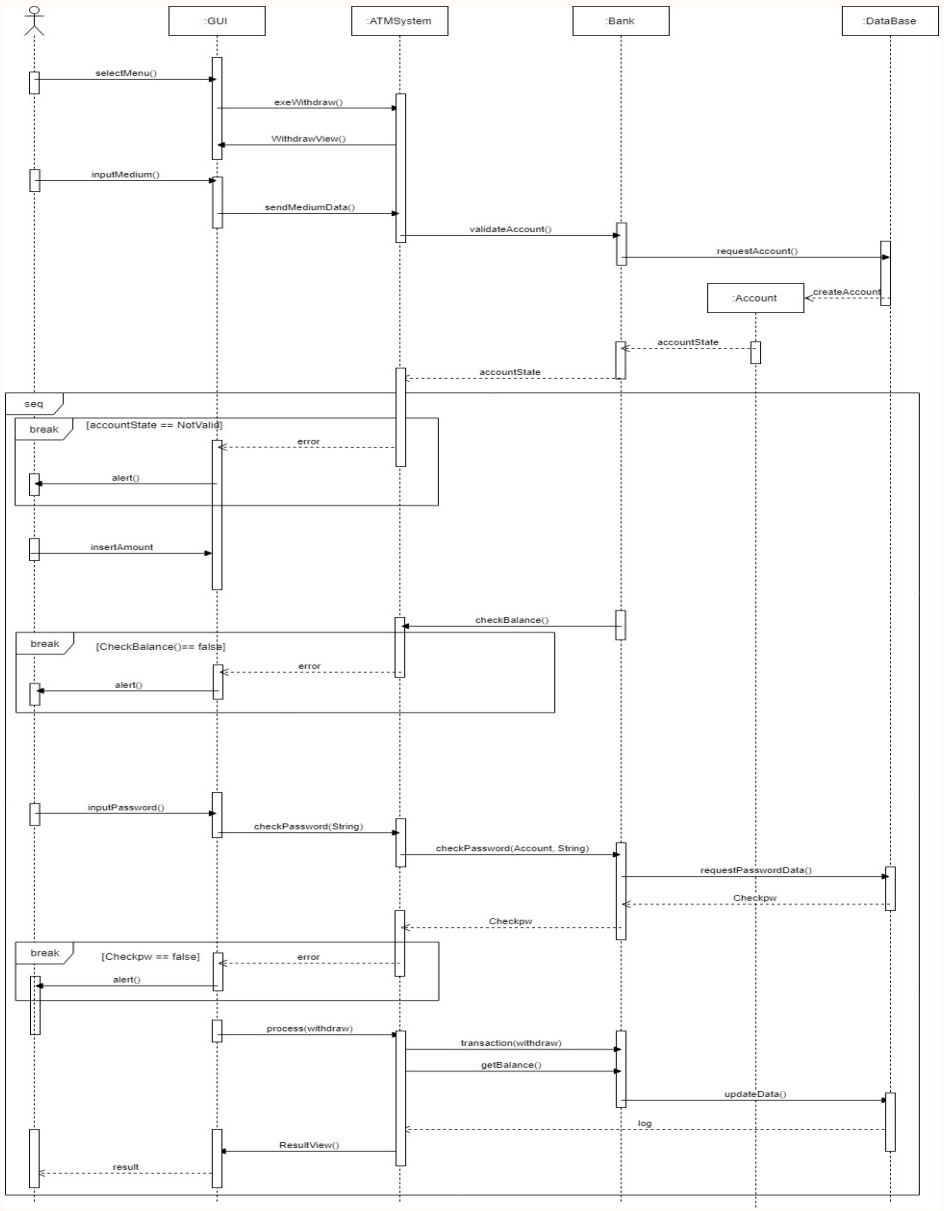
2.2 Activity 2052



Name	alert “Invalid Account”
Type	GUI
Responsibilities	계좌가 유효하지 않음을 알린다.
Cross Reference	R 3.1
Notes	계좌가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 계좌를 입력한다.

Name	alert “잔액이 부족합니다”
Type	GUI
Responsibilities	출금할 양보다 통장의 잔고가 더 적음을 알린다.
Cross Reference	R 4.4
Notes	출금할 양보다 통장의 잔고가 더 적음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	통장의 잔고보다 더 많은 돈을 입력한다.

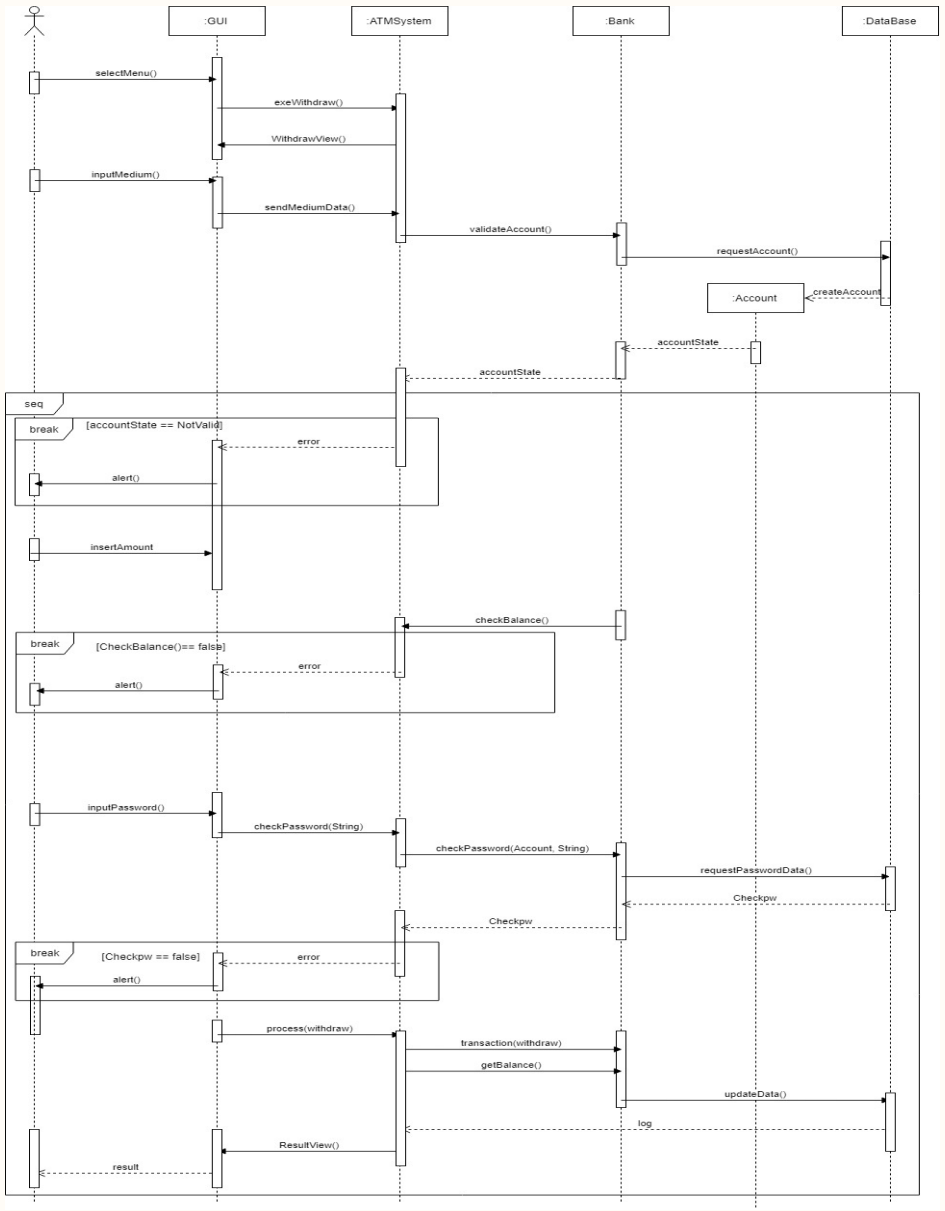
2.2 Activity 2052



Name	inputPassword
Type	GUI
Responsibilities	계좌의 비밀번호를 입력할 수 있다.
Cross Reference	R 3.2
Notes	비밀번호를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	비밀번호가 유효할 경우 출금이 계속 진행되고 결과창이 뜬다.
Pre-Condition	출금할 돈을 입력받고 잔액이 충분한 상태여야 한다.

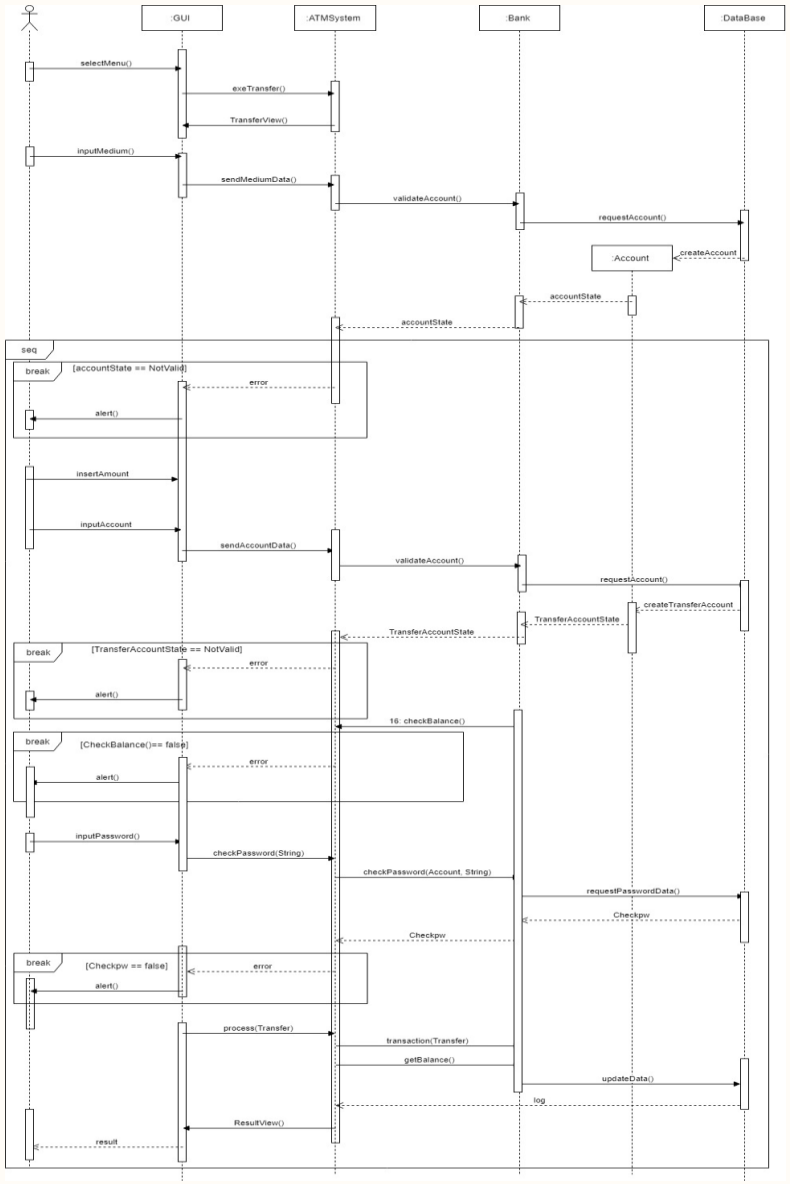
Name	alert “wrong password”
Type	GUI
Responsibilities	비밀번호가 유효하지 않음을 알린다.
Cross Reference	R 3.3
Notes	비밀번호가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 비밀번호를 입력한다.

2.2 Activity 2052



Name	ResultView
Type	GUI
Responsibilities	출금이 완료된 후 결과와 내역을 보여준다.
Cross Reference	R 6.1
Notes	작업, 금액, 잔액의 출금 내역을 보여준다.
Post-Condition	끝내기 버튼을 누를 수 있다.
Pre-Condition	유효한 비밀번호를 입력하고 출금이 계속 진행된다.

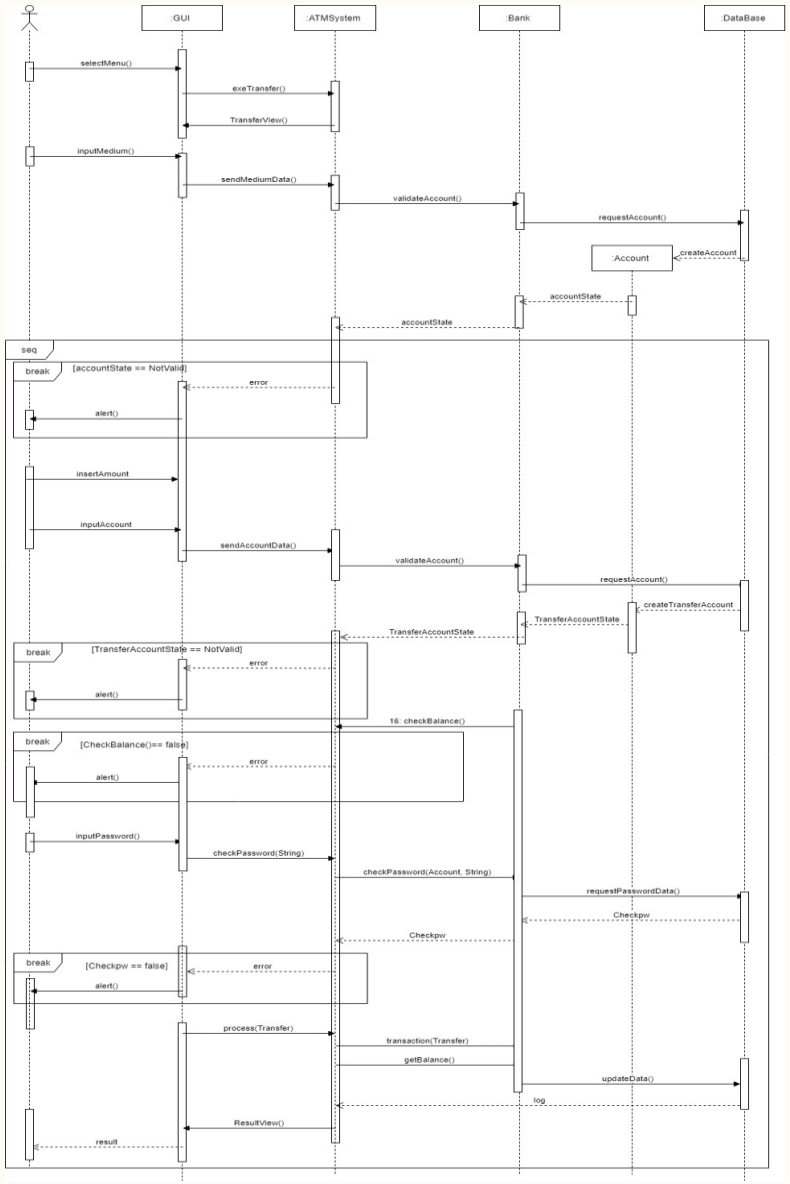
2.2 Activity 2052



Name	TransferView
Type	GUI
Responsibilities	송금을 진행하는 화면이다.
Cross Reference	R 5.3
Notes	송금을 진행하는 화면이다.
Post-Condition	계좌의 비밀번호를 입력할 수 있다.
Pre-Condition	송금 버튼을 누른 상황이어야 한다.

Name	inputMedium
Type	GUI
Responsibilities	매체의 계좌를 입력할 수 있다.
Cross Reference	R 2.1
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효할 경우 송금할 돈을 입력할 수 있다.
Pre-Condition	송금 버튼을 누른 상황이어야 한다.

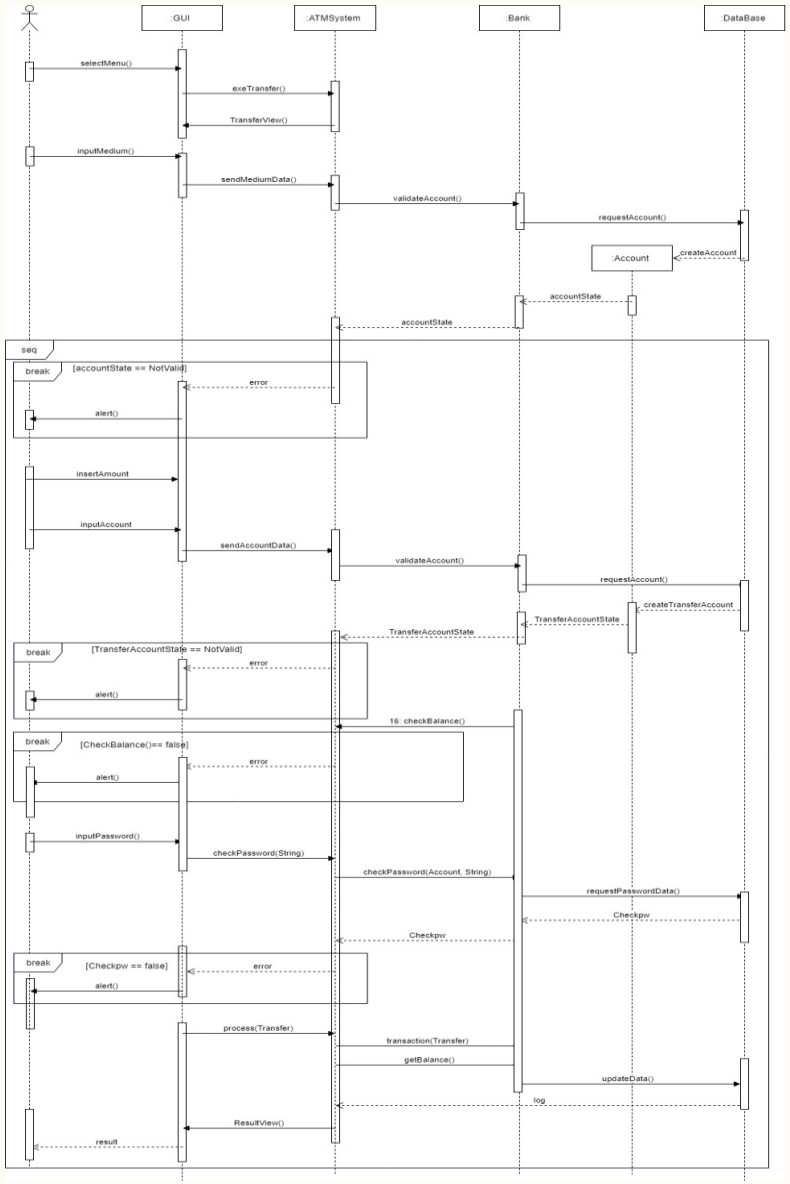
2.2 Activity 2052



Name	inputAccount
Type	GUI
Responsibilities	송금할 계좌를 입력할 수 있다.
Cross Reference	R 4.3
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효하고 송금할 돈이 잔고보다 적을 경우 비밀번호를 입력할 수 있다.
Pre-Condition	송금할 돈을 입력한 상황이어야 한다.

Name	inputPassword
Type	GUI
Responsibilities	계좌의 비밀번호를 입력할 수 있다.
Cross Reference	R 3.2
Notes	비밀번호를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	비밀번호가 유효할 경우 송금이 계속 진행되고 결과창이 뜬다.
Pre-Condition	송금할 돈을 입력받고 잔액이 충분한 상태여야 한다.

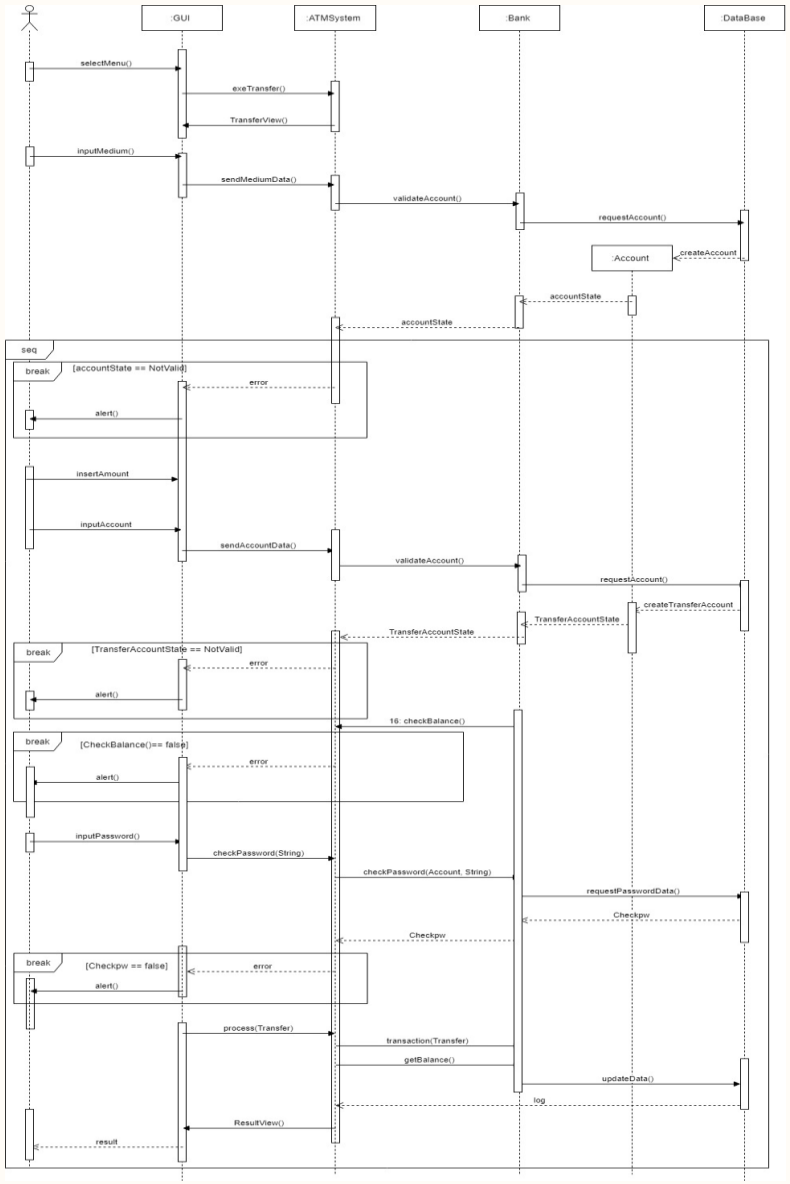
2.2 Activity 2052



Name	alert “wrong password”
Type	GUI
Responsibilities	비밀번호가 유효하지 않음을 알린다.
Cross Reference	R 3.3
Notes	비밀번호가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 비밀번호를 입력한다.

Name	alert “Invalid Account”
Type	GUI
Responsibilities	계좌가 유효하지 않음을 알린다.
Cross Reference	R 3.1
Notes	계좌가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 계좌를 입력한다.

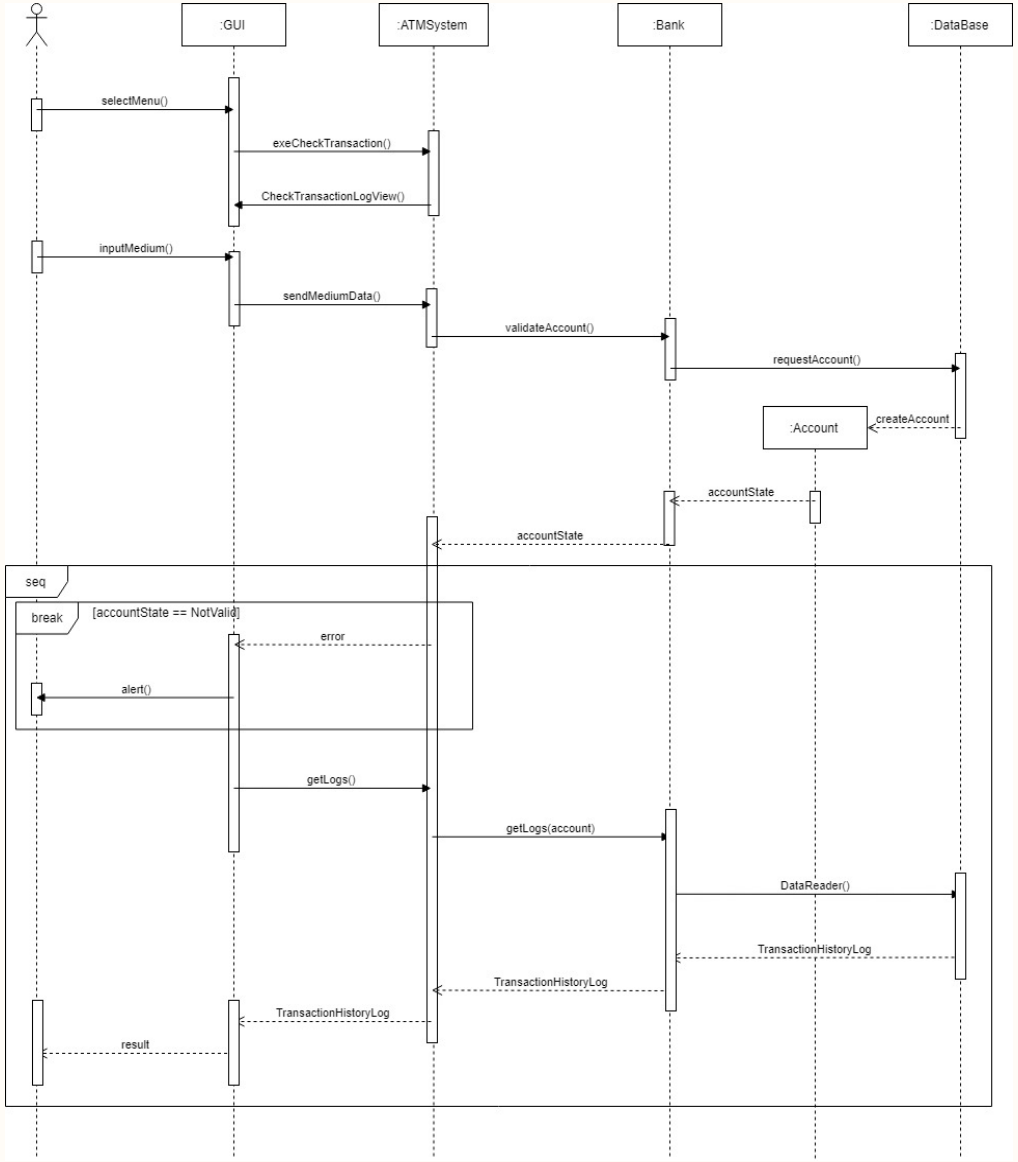
2.2 Activity 2052



Name	alert “잔액이 부족합니다”
Type	GUI
Responsibilities	송금할 양보다 통장의 잔고가 더 적음을 알린다.
Cross Reference	R 4.4
Notes	송금할 양보다 통장의 잔고가 더 적음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	통장의 잔고보다 더 많은 돈을 입력한다.

Name	ResultView
Type	GUI
Responsibilities	송금이 완료된 후 결과와 내역을 보여준다.
Cross Reference	R 6.1
Notes	작업, 금액, 잔액의 송금 내역을 보여준다.
Post-Condition	끝내기 버튼을 누를 수 있다.
Pre-Condition	유효한 비밀번호를 입력하고 송금이 계속 진행된다.

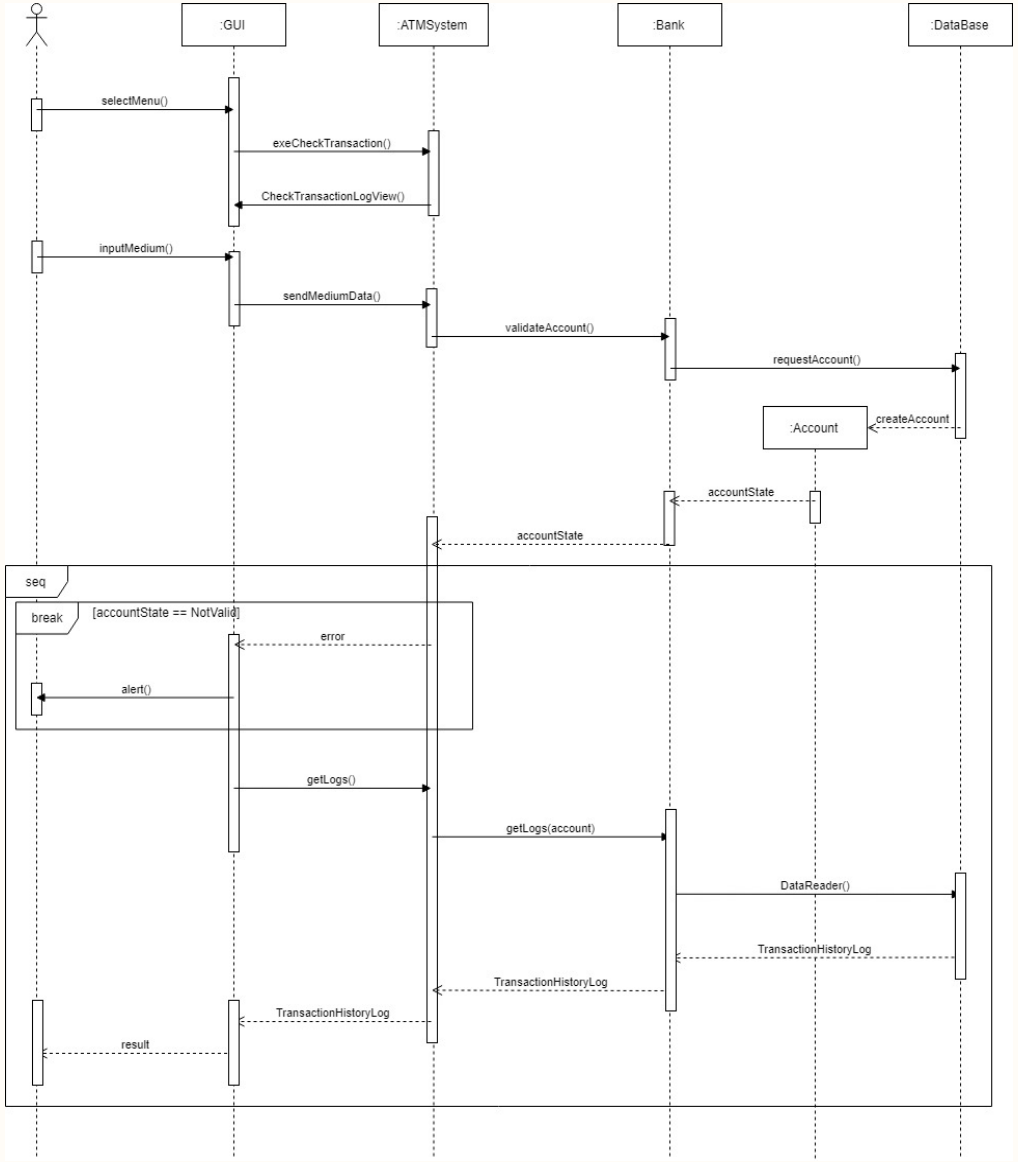
2.2 Activity 2052



Name	CheckTransacationHistoryView
Type	GUI
Responsibilities	조회를 진행하는 화면이다.
Cross Reference	R 5.4
Notes	매체를 입력할 수 있다.
Post-Condition	계좌가 유효한지 확인하고 유효할 경우 거래내역을 스크롤바로 조회할 수 있다.
Pre-Condition	조회 버튼을 누른 상황이어야 한다.

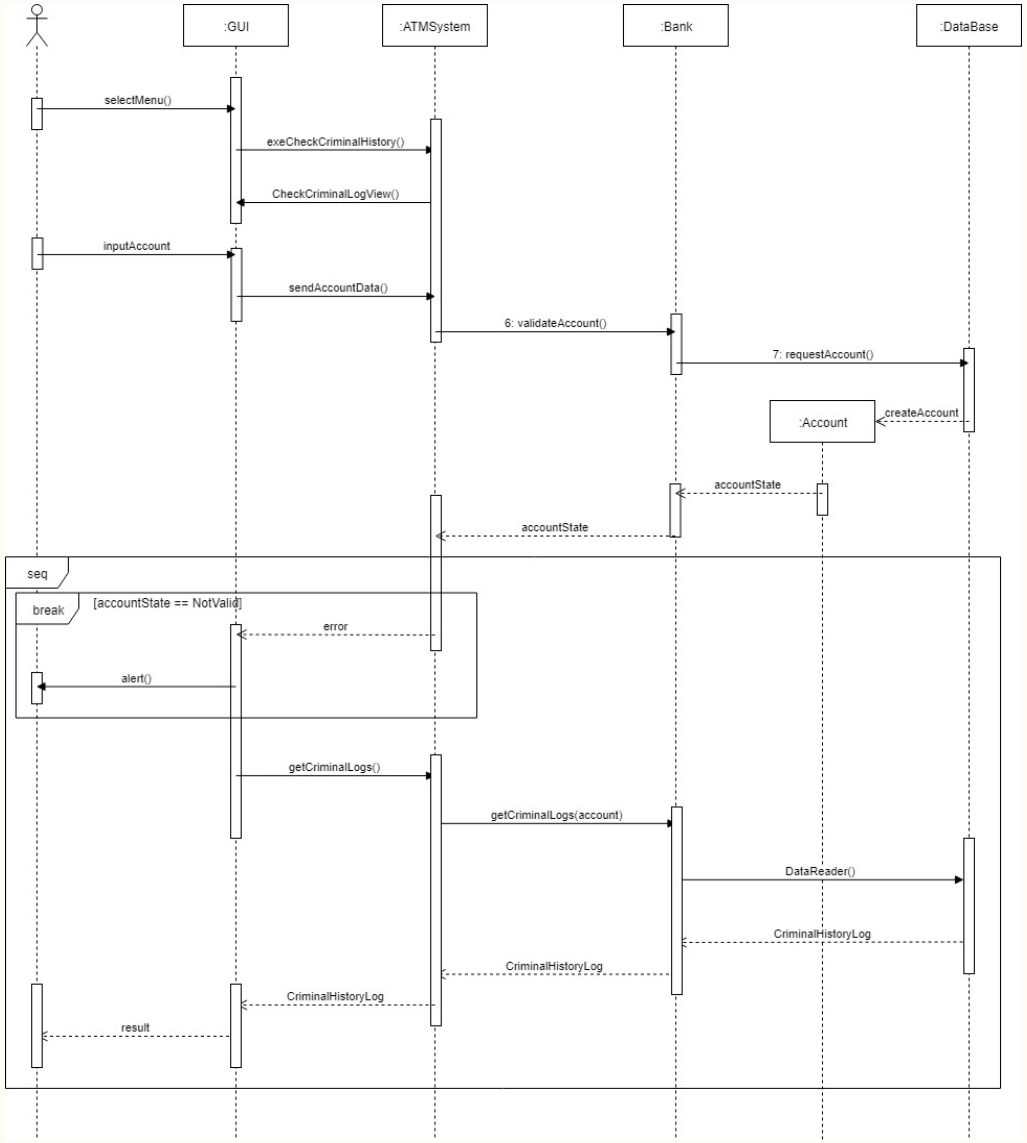
Name	inputMedium
Type	GUI
Responsibilities	매체의 계좌를 입력할 수 있다.
Cross Reference	R 2.1
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효할 경우 거래내역을 띄운다.
Pre-Condition	거래내역 조회 버튼을 누른 상황이어야 한다.

2.2 Activity 2052



Name	alert “Invalid Account”
Type	GUI
Responsibilities	계좌가 유효하지 않음을 알린다.
Cross Reference	R 3.1
Notes	계좌가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 계좌를 입력한다.

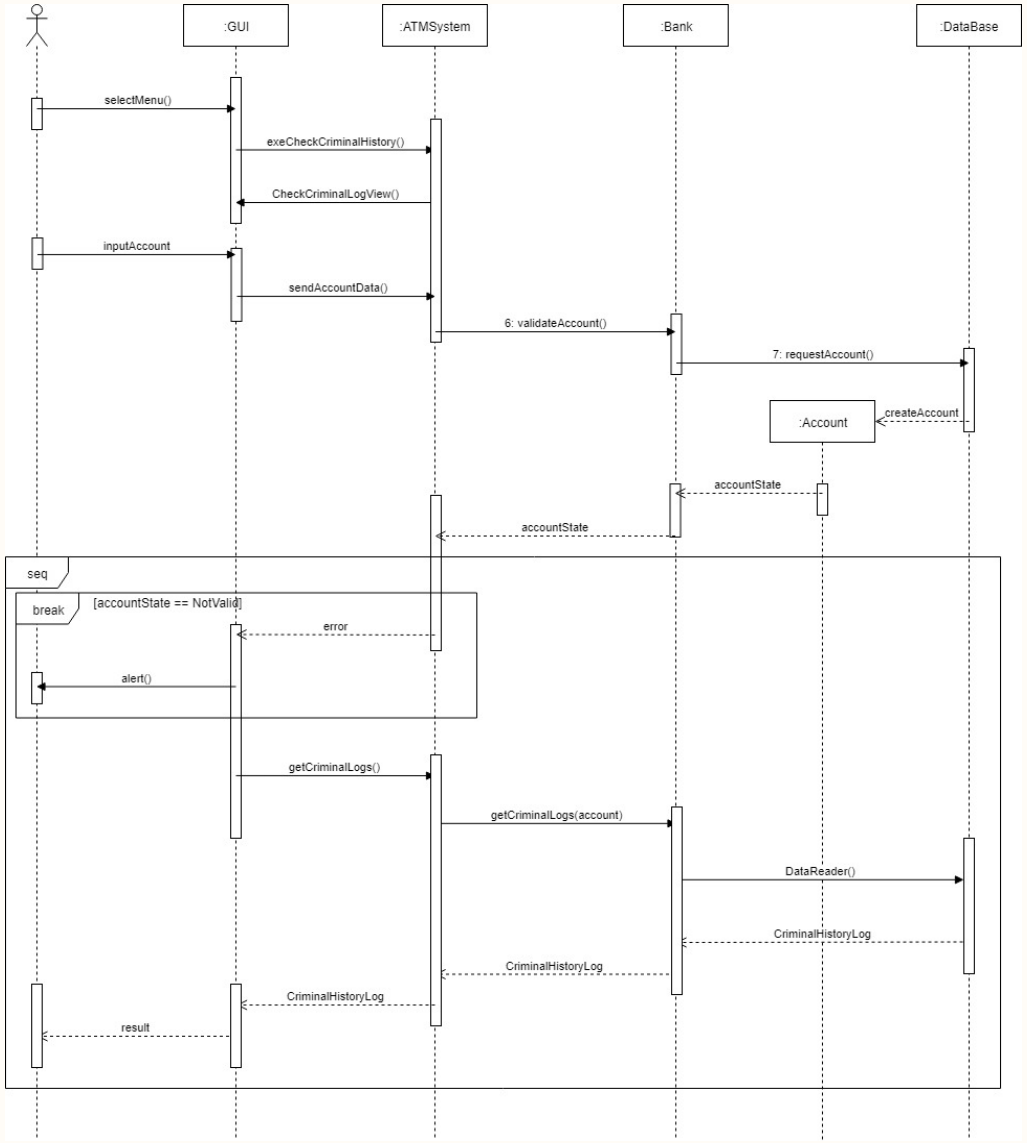
2.2 Activity 2052



Name	CheckCriminalHistoryView
Type	GUI
Responsibilities	범죄 이력 조회를 진행하는 화면이다.
Cross Reference	R 5.5
Notes	범죄 이력을 조회하고 싶은 계좌를 입력할 수 있다.
Post-Condition	계좌가 유효한지 확인하고 유효할 경우 범죄이력내역을 스크롤바로 조회할 수 있다.
Pre-Condition	범죄이력조회 버튼을 누른 상황이어야 한다.

Name	inputAccount
Type	GUI
Responsibilities	범죄 이력을 조회하고 싶은 계좌를 입력할 수 있다.
Cross Reference	R 2.1
Notes	계좌를 입력받은 뒤 유효하지 않을 경우 에러 메시지를 출력하는 화면으로 넘어간다.
Post-Condition	계좌가 유효할 경우 범죄 이력 내역을 띄운다.
Pre-Condition	범죄이력 조회 버튼을 누른 상황이어야 한다.

2.2 Activity 2052



Name	alert “Invalid Account”
Type	GUI
Responsibilities	계좌가 유효하지 않음을 알린다.
Cross Reference	R 3.1
Notes	계좌가 유효하지 않음을 알린다.
Post-Condition	확인 버튼을 누를 수 있다.
Pre-Condition	유효하지 않은 계좌를 입력한다.

2.3 Activity 2055 – Unit Test Code

```java
public class Activate {
    public int activate() {
        String flag = "";
        Scanner scanner = new Scanner(System.in);
        while(!flag.equals("N")) {
            selectMenu();
            System.out.println("Complete selectMenu()");
            do {
                System.out.println("Continue? (Y/N)");
                flag = scanner.nextLine();
            }while(!flag.equals("Y") && !flag.equals("N"));

            if(flag.equals("Y")) {
                return 1;
            }
        }
        return 2;
    }

    public void selectMenu() {

    }
}
```

```java
class ActivateTest {

    @Test
    void testActivate_1() {
        Activate Acti = new Activate();
        int x = Acti.activate();
        assertEquals(1, x);
    }

    @Test
    void testActivate_2() {
        Activate Acti = new Activate();
        int x = Acti.activate();
        assertEquals(2, x);
    }

}
```

```java
public class SelectMenu {

    public int selectMenu() {

    Scanner scanner = new Scanner(System.in);
    System.out.print("Select task progress : ");
    int menu = scanner.nextInt();

    switch(menu) {
        case 1 :
            deposit();
            return 1;
        case 2:
            withdraw();
            return 2;
        case 3:
            transfer();
            return 3;
        case 4:
            checkTransactionHistory();
            return 4;
        case 5:
            checkCriminalHistory();
            return 5;
        default:
            System.out.println("invalid access");
            return 6;
        }
    }
```

```java
class SelectMenuTest {

    @Test
    void testSelectMenu_1() {
        SelectMenu Sele = new SelectMenu();
        int result = Sele.selectMenu();
        assertEquals(1, result);
    }

    @Test
    void testSelectMenu_2() {
        SelectMenu Sele = new SelectMenu();
        int result = Sele.selectMenu();
        assertEquals(2, result);
    }

    @Test
    void testSelectMenu_3() {
        SelectMenu Sele = new SelectMenu();
        int result = Sele.selectMenu();
        assertEquals(3, result);
    }

    @Test
    void testSelectMenu_4() {
        SelectMenu Sele = new SelectMenu();
        int result = Sele.selectMenu();
        assertEquals(4, result);
    }
```

```java
@Test
void testSelectMenu_5() {
    SelectMenu Sele = new SelectMenu();
    int result = Sele.selectMenu();
    assertEquals(5, result);
}

@Test
void testSelectMenu_6() {
    SelectMenu Sele = new SelectMenu();
    int result = Sele.selectMenu();
    assertEquals(6, result);
}
```

2.3 Activity 2055 – Unit Test Code

```java
public class FindAccount {
    public int findAccount(String accountNum) {
        //for (Bank bank : banks) {
            Bank bank = new Bank();
            Account ret = bank.validateAccount(accountNum);
            if (ret != null) {
                return 1;
            }//}

        System.out.println("invalid account");
        return 2;
    }
}
```

```java
class FindAccountTest {

    @Test
    void testFindAccount_1() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("1234");
        assertEquals(1, result);
    }

    @Test
    void testFindAccount_2() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("1256");
        assertEquals(2, result);
    }

    @Test
    void testFindAccount_3() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("12566");
        assertEquals(2, result);
    }

}
```


```java
public class FindAccount {
    public int findAccount(String accountNum) {
        //for (Bank bank : banks) {
            Bank bank = new Bank();
            Account ret = bank.validateAccount(accountNum);
            if (ret != null) {
                return 1;
            }//}

        System.out.println("invalid account");
        return 2;
    }
}
```

```java
class FindAccountTest {

    @Test
    void testFindAccount_1() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("1234");
        assertEquals(1, result);
    }

    @Test
    void testFindAccount_2() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("1256");
        assertEquals(2, result);
    }

    @Test
    void testFindAccount_3() {
        FindAccount Fa = new FindAccount();
        int result = Fa.findAccount("12566");
        assertEquals(2, result);
    }

}
```

2.3 Activity 2055 – Unit Test Code

```java
public class InputAccountNum {
    public int inputAccount() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("input AccountNum: ");
        String accountNum = scanner.nextLine();
        Account ret = new Account();
        ret = findAccount(accountNum);

        if(ret != null) {
            return 1;
        }else {
            return 2;
        }
    }
```

```java
class InputAccountNumTest {

    @Test
    void testInputAccount_1() {
        InputAccountNum IA = new InputAccountNum();
        int result = IA.inputAccount();
        assertEquals(1, result);
    }

    @Test
    void testInputAccount_2() {
        InputAccountNum IA = new InputAccountNum();
        int result = IA.inputAccount();
        assertEquals(2, result);
    }
}
```

```java
public class Deposit {
    public int deposit() {
        Account src = inputMedium(); //src==null 인경우 exception
        if (src == null) { //!
            return 1;
        }

        //Bank bank = src.getBank();
        String pw = inputPassword();
        if (!pw.equals("1111")) { //!
            return 2;
        }

        int amount = insertCash();


        System.out.println("bank.transaction complete " + amount);
        //bank.transaction(src,amount,"deposit");

        System.out.println("deposit complete");
        //this.printReceipt("deposit",amount,bank.getBalance(src));

        return 3; //!
    }
```

```java
class DepositTest {

    @Test
    void testDeposit_1() {
        Deposit D = new Deposit();
        int result = D.deposit();
        assertEquals(1, result);
    }

    @Test
    void testDeposit_2() {
        Deposit D = new Deposit();
        int result = D.deposit();
        assertEquals(2, result);
    }

    @Test
    void testDeposit_3() {
        Deposit D = new Deposit();
        int result = D.deposit();
        assertEquals(3, result);
    }
```

2.3 Activity 2055 – Unit Test Code

```java
public int withdraw(){
    Account src = inputMedium();
    if (src == null) {
        return 1; //! 1
    }
//Bank bank = src.getBank();

    String pw = inputPassword();
    if (!pw.equals("1111")) {
        return 2; //! 2
    }

    int amount = insertAmount();

    //if (!bank.transaction(src,amount * -1,"withdraw")) {
    if(!transaction(amount)){
        return 3; //! 3
    }


    //this.printReceipt("withdraw",amount,bank.getBalance(src));

    System.out.println("Complete withdraw " + amount);
    return 4; //!
    }
```

```java
@Test
void testWithdraw_1() {
    Withdraw W = new Withdraw();
    int result = W.withdraw();
    assertEquals(1, result);
}

@Test
void testWithdraw_2() {
    Withdraw W = new Withdraw();
    int result = W.withdraw();
    assertEquals(2, result);
}

@Test
void testWithdraw_3() {
    Withdraw W = new Withdraw();
    int result = W.withdraw();
    assertEquals(3, result);
}

@Test
void testWithdraw_4() {
    Withdraw W = new Withdraw();
    int result = W.withdraw();
    assertEquals(4, result);
}
```

```java
public int transfer() {
    Account src = inputMedium();
    if (src == null) {
        return 1; //! 1
    }
    //Bank srcBank = src.getBank();
    String pw = inputPassword();
    if (!pw.equals("1111")) {
        return 2; //!2
    }

    Account des = inputAccount();
    if (des == null) {
        return 3; //!3
    }

    //Bank desBank = des.getBank();

    int amount = insertAmount();
    //if (!srcBank.transaction(src,amount * -1,"transfer to " + des.getAccountNum())) {
    if(!transaction(amount)){
        return 4; //!4
    }

    //desBank.transaction(des,amount,"transfer from " + src.getAccountNum());

    //this.printReceipt("transfer",amount,srcBank.getBalance(src));
    System.out.println("Complete Transfer");
    return 5;
```

```java
void testTransfer_1() {
    Transfer T = new Transfer();
    int result = T.transfer();
    assertEquals(1, result);
}

@Test
void testTransfer_2() {
    Transfer T = new Transfer();
    int result = T.transfer();
    assertEquals(2, result);
}

@Test
void testTransfer_3() {
    Transfer T = new Transfer();
    int result = T.transfer();
    assertEquals(3, result);
}

@Test
void testTransfer_4() {
    Transfer T = new Transfer();
    int result = T.transfer();
    assertEquals(4, result);
}

@Test
void testTransfer_5() {
    Transfer T = new Transfer();
    int result = T.transfer();
```

2.3 Activity 2055 – Unit Test Code

```java
public class CheckTransactionHistory {
    public int checkTransactionHistory() {
            Account src = inputMedium();
            if (src == null) {
                return 1; //!1
            }
            //Bank bank = src.getBank();
            //bank.showLogs(src);
            System.out.println("Complete CheckTransactionHistory");
            return 2;//!2
    }
```

```java
class CheckTransactionHistoryTest {

    @Test
    void testCheckTransactionHistory_1() {
        CheckTransactionHistory CT = new CheckTransactionHistory();
        int result = CT.checkTransactionHistory();
        assertEquals(1, result);
    }

    @Test
    void testCheckTransactionHistory_2() {
        CheckTransactionHistory CT = new CheckTransactionHistory();
        int result = CT.checkTransactionHistory();
        assertEquals(2, result);
    }
}
```

```java
public class CheckCriminalHistory {
    public int checkCriminalHistory() {
            Account src = inputAccount();
             if (src == null) {
                 return 1; //!1
             }
             //Bank bank = src.getBank();
             //bank.showLogs(src);
             System.out.println("Complete CriminalHistory");
             return 2;//!2
    }

    public Account inputAccount() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("input AccountNum: ");
        String accountNum = scanner.nextLine();
        Account ret = new Account();
        ret = findAccount(accountNum);

        return ret;
    }
```

```java
@Test
void testCheckCriminalHistory_1() {
    CheckCriminalHistory CT = new CheckCriminalHistory();
    int result = CT.checkCriminalHistory();
    assertEquals(1, result);
}

@Test
void testCheckCriminalHistory_2() {
    CheckCriminalHistory CT = new CheckCriminalHistory();
    int result = CT.checkCriminalHistory();
    assertEquals(2, result);
}
```

2.3 Activity 2055 – Unit Test Code

```java
public class _FileReader {

    public int _fileReader() {
        //File file = new File();
        System.out.println("File");
       // ArrayList<String> lines = new ArrayList<>();
        System.out.println("ArrayList");
        try {
            //FileReader reader = new FileReader(file);
            System.out.println("FileReader");
            //BufferedReader bufferedReader = new BufferedReader(reader);
            System.out.println("bufferedReader");
            //String line;
            //while((line = bufferedReader.readLine()) != null) {
            //    lines.add(line);
            //}
            System.out.println("add line");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return 1;
    }
```

```java
class _FileReaderTest {

    @Test
    void test_fileReader() {
        _FileReader F = new _FileReader();
        int result = F._fileReader();
        assertEquals(1, result);
    }

}
```

```java
public class ValidateAccount {
    public Account validateAccount(boolean exist) {
        //String path = this.path + "/" + accountNum;
        //File file = new File(path);
        if(exist) {
            Account ret = new Account();
            return ret;

        } else {
            return null;

        }
    }
}
```

```java
class ValidateAccountTest {

    @Test
    void testValidateAccount_1() {
        ValidateAccount va = new ValidateAccount();
        assertNotNull(va.validateAccount(true));
    }

    @Test
    void testValidateAccount_2() {
        ValidateAccount va = new ValidateAccount();
        Account result = va.validateAccount(false);
        assertEquals(null, result);
    }

}
```

2.3 Activity 2055 – Unit Test Code

```
ublic class Checkpassword {
    public boolean checkPassword(String password) {
        //String path = this.path + "/" + account.getAccountNum() + "/password.txt";
        //String pw = this.fileReader(path).get(0);
        String pw = "1111";
        return pw.equals(password);
    }
}
```

```
class CheckpasswordTest {

    @Test
    void testCheckPassword_1() {
        Checkpassword cp = new Checkpassword();
        boolean result = cp.checkPassword("1234");
        assertEquals(false, result);
    }

    @Test
    void testCheckPassword_2() {
        Checkpassword cp = new Checkpassword();
        assertTrue(cp.checkPassword("1111"));
    }

}
```

```
public class GetBalance {
    public int getBalance(int size) {
        //ArrayList<String> logs = this.getLogs(account);
        if (size == 0) {
            return 1;
        }
        //String lastLog = logs.get(logs.size()-1);
        //String balance = lastLog.split(";")[3];
        //return Integer.parseInt(balance);
        return 2;
    }
```

```
class GetBalanceTest {

    @Test
    void testGetBalance_1() {
        GetBalance gb = new GetBalance();
        int result = gb.getBalance(0);
        assertEquals(1, result);
    }

    @Test
    void testGetBalance_2() {
        GetBalance gb = new GetBalance();
        int result = gb.getBalance(3);
        assertEquals(2, result);
    }

}
```

2.3 Activity 2055 – Unit Test Code

```java
public class Transaction {

    public boolean transaction(int amount) {
        //int balance = this.getBalance(account);
        if(amount < 0) {
            return false;
        }
        //Date cur = new Date();
        //SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        //String date = dateFormat.format(cur);
        //String log = date + ";" + msg + ";" + amount + ";" + (balance + amount) + "\n";
        //File logFile = new File(this.path + "/" + account.getAccountNum() + "/log.txt");
        try {
            //FileWriter writer = new FileWriter(logFile,true);
            //writer.write(log);
            //writer.flush();
            //writer.close();
            System.out.println("Filewrite");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return true;
    }
}
```

```java
class TransactionTest {

    @Test
    void testTransaction_1() {
        Transaction t = new Transaction();
        assertEquals(false, t.transaction(-1000));
    }

    @Test
    void testTransaction_2() {
        Transaction t = new Transaction();
        assertTrue(t.transaction(1000));
    }

}
```

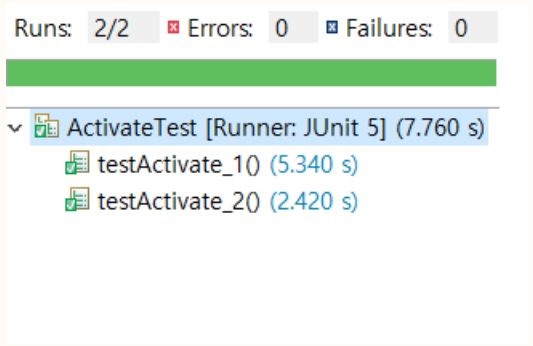
```java
public int getLogs() {
    //String path = this.path + "/" + account.getAccountNum() + "/log.txt";
    System.out.println("path");
    //return this.fileReader(path);
    System.out.println("fileReader");
    return 1;
}
```

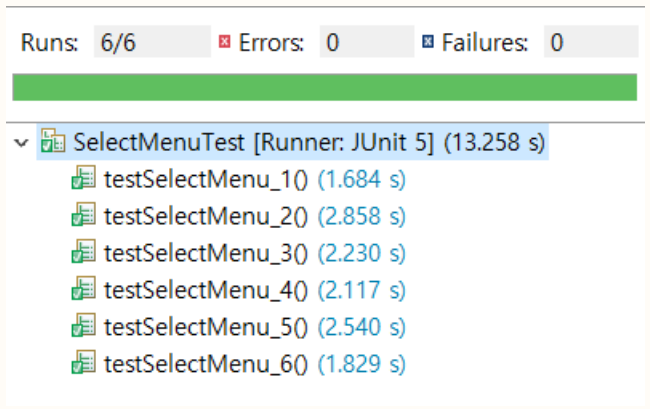
```java
class GetlogsTest {

    @Test
    void testGetLogs() {
        Getlogs gl = new Getlogs();
        int result = gl.getLogs();
        assertEquals(1, result);
    }

}
```

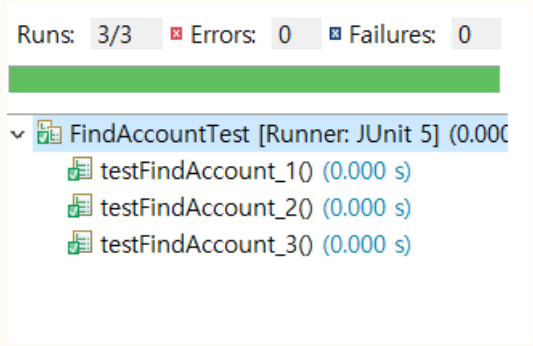
```java
public class ShowLogs {
    public int showLogs() {
            //for (String log : this.getLogs(account)) {
            //    System.out.println(log);
            //}
            System.out.println("showlogs");
            return 1;
        }
}
```

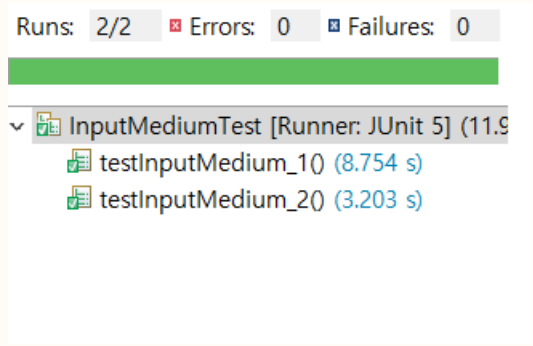
```java
class ShowLogsTest {

    @Test
    void testShowLogs() {
        ShowLogs slt = new ShowLogs();
        int result = slt.showLogs();
        assertEquals(1, result);
    }

}
```

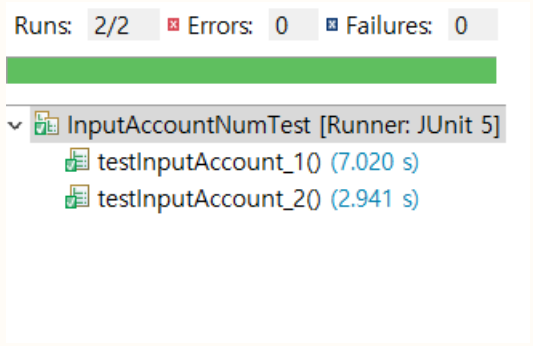
2.4 Activity 2061 – Unit Testing

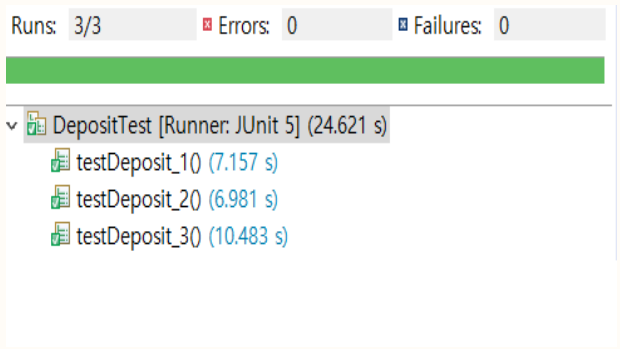


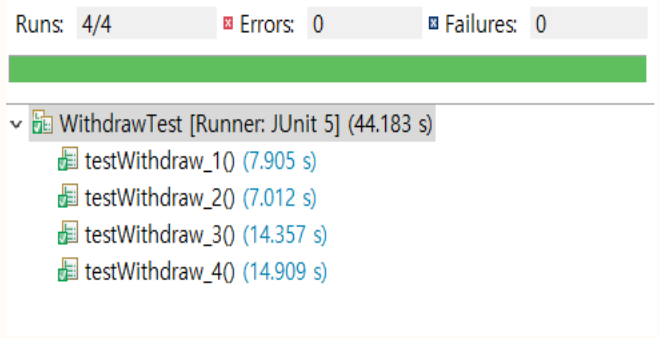


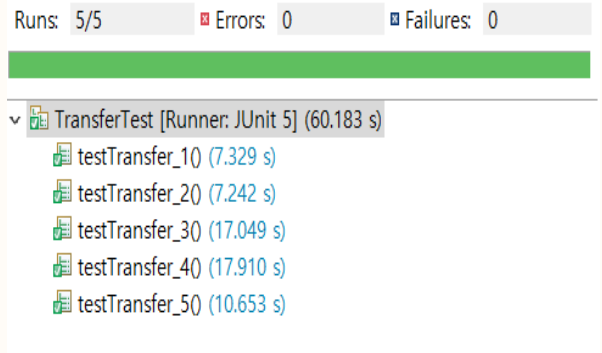




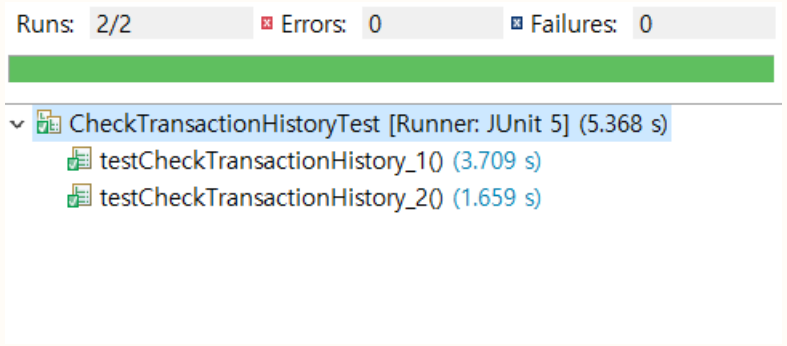


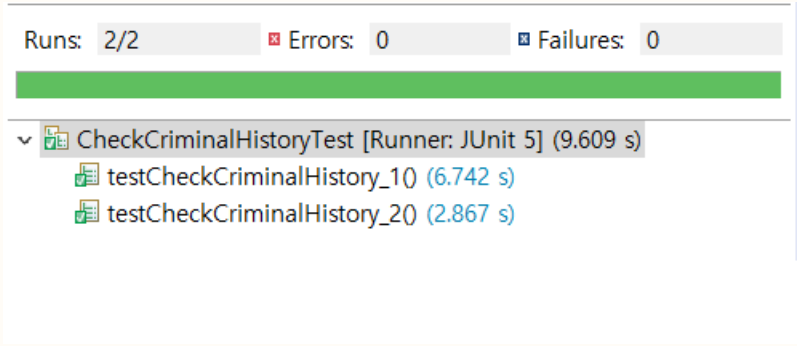


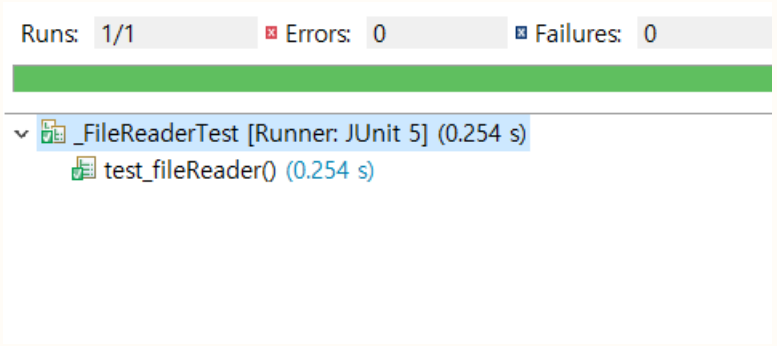


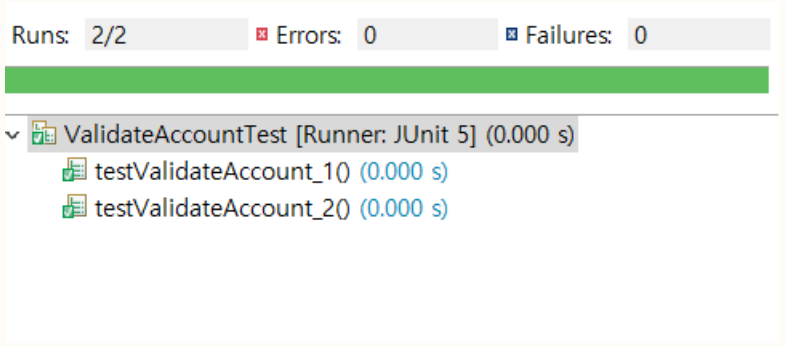


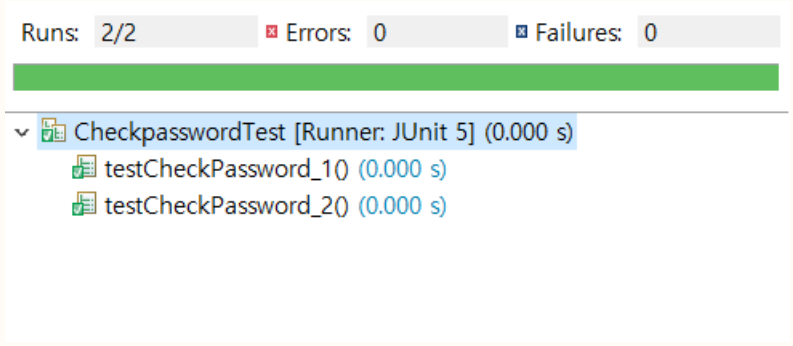
2.4 Activity 2061 – Unit Testing

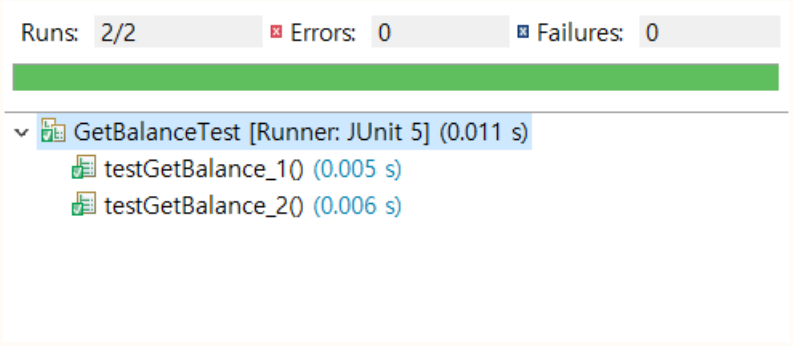




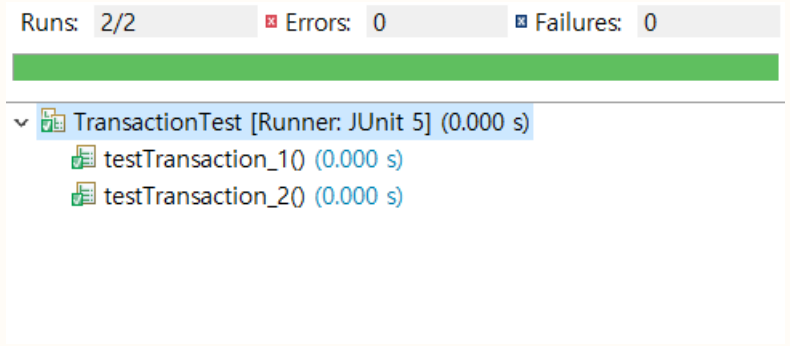


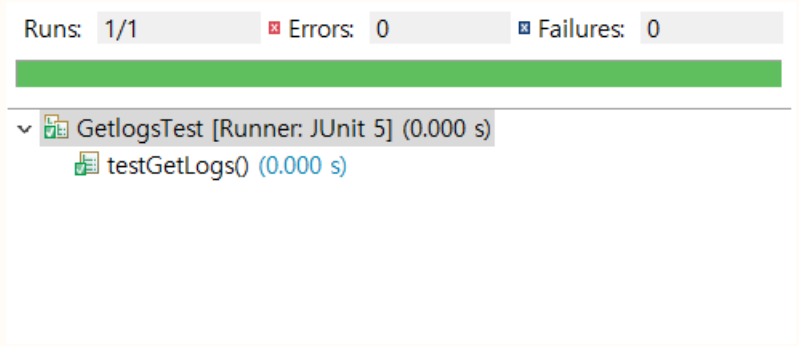


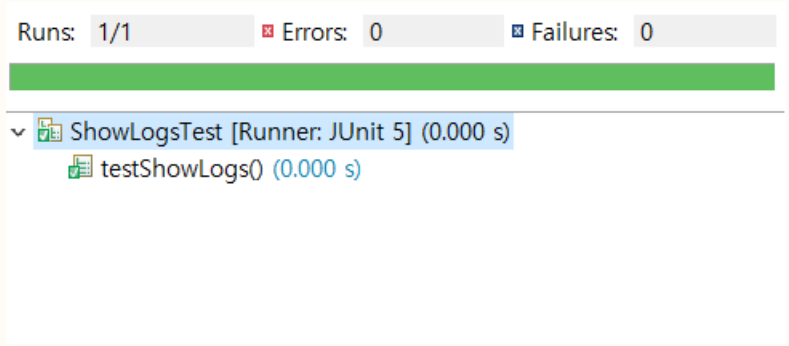




2.4 Activity 2061 – Unit Testing







2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
1_1	selectMenuTest	Deposit을 눌렀을때 정상적으로 메뉴선택이 되는지 확인한다.	R 1.1	P
1_2	selectMenuTest	Withdraw을 눌렀을때 정상적으로 메뉴선택이 되는지 확인한다.	R 1.1	P
1_3	selectMenuTest	Transfer을 눌렀을때 정상적으로 메뉴선택이 되는지 확인한다.	R 1.1	P
1_4	selectMenuTest	CheckTransactionHistory을 눌렀을때 정상적으로 메뉴선택이 되는지 확인한다.	R 1.1	P
1_5	selectMenuTest	CheckCriminalHistory을 눌렀을때 정상적으로 메뉴선택이 되는지 확인한다.	R 1.1	P
1_6	selectMenuTest	오류처리가 되었는지 확인한다.	R 1.1	P

2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
2_1	inputMediumTest	매체를 입력하는 화면이떴을 때 매체를 입력했을 때 제대로 진행되는지 확인한다.	R 2.1	P
3_1_1	validateAccountTest	알맞은 계좌를 입력했을때 제대로 진행하는지 확인한다.	R 3.1	P
3_1_2	validateAccountTest	잘못된 계좌를 입력했을때 오류처리가 되었는지 확인한다.	R 3.1	P
3_2	inputPasswordTest	비밀번호를 입력하는 화면이 떴을 때 알맞은 비밀번호를 입력했을때 제대로 진행되는지 확인한다.	R 3.2	P
3_3	checkPasswordTest	비밀번호를 입력하는 화화면 떴을 때 잘못된 비밀번호를 입력했을때 오류처리가 되었는지 확인한다.	R 3.3	P

2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
4_1	insertMoneyTest	돈을 입력하고 그만큼 제대로 계산되는지 확인한다.	R 4.1	P
4_2	insertAmountTest	거래하거나 출금할 양을 입력하고 그만큼 정확히 계산되는지 확인한다.	R 4.2	P
4_3	inputAccountTest	계좌를 입력하는 화면이 떴을 때 제대로 입력되고 넘어가는지 확인한다.	R 4.3	P
4_4_1	CalculateBalance	1.당행 간의 거래 시 수수료가 정확히 계산된다. 2. 잔고를 넘지 않는 선에서 거래를 할 때 제대로 진행된다.	R 4.4	P
4_4_2	CalculateBalance	1.타행 간의 거래 시 수수료가 정확히 계산된다. 2. 잔고를 넘지 않는 선에서 거래를 할 때 제대로 진행된다.	R 4.4	P
4_4_3	CalculateBalance	1.당행 간의 거래 시 수수료가 정확히 계산된다. 2. 잔고를 넘어 거래를 할 때 오류처리가 된다.	R 4.4	P
4_4_4	CalculateBalance	1.타행 간의 거래 시 수수료가 정확히 계산된다. 2. 잔고를 넘어 거래를 할 때 오류처리가 된다.	R 4.4	P

2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
5_1_1	exeDepositTest	1. 메뉴 선택창에서 Deposit을 선택했을 때 Deposit화면이 뜨는지 확인한다.	R 5.1	P
5_1_2	exeDepositTest	2. Deposit 화면이 뜬 후 알맞은 매체를 입력한뒤 예금할 돈을 입력하는 화면이 제대로 뜨는지 확인한다.	R 5.1	P
5_1_3	exeDepositTest	3. 알맞은 비밀번호를 입력한 뒤 Deposit이 완료된 후 나오는 결과창에서 돈이 정확히 잘 입금되었는지 확인한다.	R 5.1	P
5_2_1	exeWithdrawTest	1. 메뉴 선택창에서 Withdraw을 선택했을 때 Withdraw화면이 뜨는지 확인한다.	R 5.2	P
5_2_2	exeWtihdrawTest	2. Withdraw 화면이 뜬 후 알맞은 매체를 입력한뒤 출금할 돈을 입력하는 화면에 제대로 뜨는지 확인한다.	R 5.2	P
5_2_3	exeWithdrawTest	3. 알맞은 비밀번호를 입력한 뒤 Withdraw이 완료된 후 나오는 결과창에서 돈이 정확히 잘 출금되었는지 확인한다.	R 5.2	P

2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
5_3_1	exeTransferTest	1. 메뉴 선택창에서 Transfer을 선택했을 때 Transfer화면이 뜨는지 확인한다.	R 5.3	P
5_3_2	exeTransferTest	2. Transfer 화면이 뜬 후 알맞은 매체를 입력한뒤 거래할 돈을 입력하는 화면이 제대로 뜨는지 확인한다.	R 5.3	P
5_3_3	exeTransferTest	3. 거래할 돈을 입력하고 송금할 계좌를 입력하는 화면이 제대로 뜨는지 확인한다.	R 5.3	P
5_3_4	exeTransferTest	4. 알맞은 비밀번호를 입력한 뒤 Transfer이 완료된 후 나오는 결과창에서 돈이 정확히 잘 송금되었는지 확인한다.	R 5.3	P
5_4_1	exeCheckTransactionHistoryTest	1. 메뉴 선택창에서 CheckTransactionHistory을 선택했을 때 CheckTransactionHistory화면이 뜨는지 확인한다.	R 5.4	P
5_4_2	exeCheckTransactionHistoryTest	2. CheckTransactionHistory 화면이 뜬 후 알맞은 매체를 입력한뒤 거래 내역이 제대로 뜨는지 확인한다.	R 5.4	P

2.5 Activity 2063 - System Testing

Test Number	Test 항목	Description	System Function	Pass/Fail
5_5_1	exeCheckCriminal HistoryTest	1. 메뉴 선택창에서 CheckCriminalHistory을 선택했을 때 CheckCriminalHistory화면이 뜨는지 확인한다.	R 5.5	P
5_5_2	exeCheckCriminal HistoryTest	2. CheckCriminalHistory 화면이 뜬 후 알맞은 계좌를 입력한뒤 거래 내역이 제대로 뜨는지 확인한다.	R 5.5	P
6_1_1	printResultTest	1. 입금이 끝난 뒤 내역이 제대로 화면에 출력되는지 확인한다.	R 6.1	P
6_1_2	printResultTest	2. 출금이 끝난 뒤 내역이 제대로 화면에 출력되는지 확인한다.	R 6.1	P
6_1_3	printResultTest	3. 송금이 끝난 뒤 내역이 제대로 화면에 출력되는지 확인한다.	R 6.1	P

3. Q & A