Software Modeling & Analysis

[OOPT stage 2050 & 2060]

No.	T7
Subject	Software Modeling & Analysis
Professor	JUNBEOM YOO
Team Member	201615007 문기태
	201410621 한상민

Chart.

Activity 2051. Implement Class & Methods Definitions

Type	Class
Name	Controller
Purpose	ATM의 기능을 사용자가 조정하고 사용하기 위한 클래스
Overview(Class)	사용자가 ATM의 모든 클래스와 Method를 사용하기 위한 수단으로 GUI로 구현된다.
Cross Reference	R1.2,1.3
Exceptional Courses of Events	N/A

Type	Class
Name	Send
Purpose	송금기능을 수행하는 클래스
Overview(Class)	사용자와 송금대상의 계좌정보를 입력 받아 송금을 수행한다.
Cross Reference	R2.1, 1.3
Exceptional Courses of Events	송금/수수료액이 한도를 넘어가거나 잔액보다 많으면 송금기능을 수행하지 않는다.

Type	Class
Name	Withdraw
Purpose	출금기능을 수행하는 클래스
Overview(Class)	사용자가 원하는 액수만큼 출금을 수행한다.
Cross Reference	R2.2, 1.3
Exceptional Courses of Events	출금/수수료액이 한도를 넘어가거나 잔액보다 많으면 송금기능을 수행하지 않는다.

Type	Class
Name	Deposit
Purpose	입금기능을 수행하는 클래스
Overview(Class)	사용자가 계좌에 입금할 돈에 대한 입금을 수행한다.
Cross Reference	R2.3
Exceptional Courses of Events	N/A

Type	Class
Name	Check Remain
Purpose	잔액조회기능을 수행하는 클래스
Overview(Class)	사용자의 계좌의 잔액을 확인해주는 기능을 수행한다.
Cross Reference	R2.4
Exceptional Courses of Events	N/A

Type	Class
Name	Payback
Purpose	환급혜택을 제공하는 클래스
Overview(Class)	사용자 계좌의 사용횟수를 조회하여 환급혜택의 유무를 결정하고 해당되는 사용자에게 환급 혜택을 제공한다.
Cross Reference	R3.2
Exceptional Courses of Events	사용자의 이용횟수가 환급조건에 해당되지 않으면 기능을 수행하지 않는다.

Type	Class
Name	Commission
Purpose	수수료를 책정하는 클래스
Overview(Class)	송금/ 출금 기능 수행 시 필요한 수수료를 시간을 통해 책정하여 부과하는 역할을 한다.
Cross Reference	R1.4
Exceptional Courses of Events	N/A

Type	Class
Name	Statement
Purpose	영수증과 관련된 일을 수행하는 클래스
Overview(Class)	잔액조회를 제외한 모든 거래가 종료된 후에 거래내용을 담은 영수증을 제공하는 역할을 한다.
Cross Reference	R3.1
Exceptional Courses of Events	사용자가 영수증 출력을 원하지 않을 경우에는 기능을 수행하지 않는다.

Type	Class
Name	Bank
Purpose	계좌의 클래스를 담고 있는 클래스
Overview(Class)	은행을 이용하는 고객의 계좌정보들의 파일과 은행의 정보를 담고 있다.
Cross Reference	R1.1
Exceptional Courses of Events	N/A

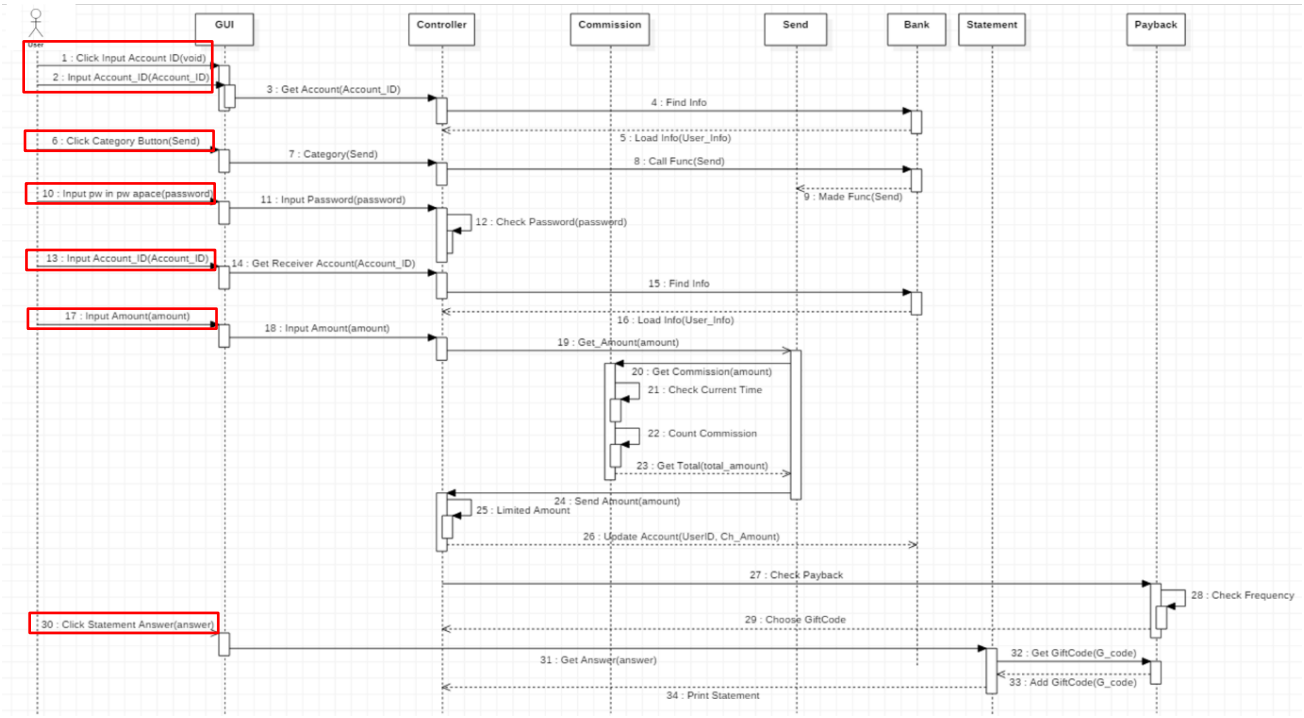
Type	Class
Name	Account
Purpose	계좌정보를 담고 있는 클래스
Overview(Class)	각 고객의 계좌에 대한 모든 정보를 담고 있는 파일이다.
Cross Reference	N/A
Exceptional Courses of Events	N/A

Type	Method
Name	Get Account()
Purpose	계좌의 아이디를 입력하는 Method다.
Overview(Class)	사용자의 아이디를 입력 받는 Method를 의미한다.
Cross Reference	N/A
Input(Method)	Account_id (사용자의 계좌 정보): Integer
Output(Method)	Void
Abstract Operation(Method)	-
Exceptional Courses of Events	N/A

Type	Method
Name	Check Password()
Purpose	비밀번호가 일치하는지 확인하는 Method다.
Overview(Class)	입력한 비밀번호가 입력한 계좌정보에 해당되는 비밀번호인지 확인한다.
Cross Reference	R1.2
Input(Method)	Password(비밀번호): Integer
Output(Method)	Boolean(비밀번호 일치여부)
Abstract Operation(Method)	-
Exceptional Courses of Events	비밀번호가 일치하지 않으면 다시 Input Password로 돌아간다..

Type	Method
Name	Get Answer()
Purpose	영수증 출력의사를 묻고 영수증 출력을 판단하는 Method다.
Overview(Class)	사용자가 입력한 영수증 출력의사를 입력 받아 영수증 출력을 판단한다.
Cross Reference	R3.1
Input(Method)	Answer(출력의사여부): String[Yes | No]
Output(Method)	Boolean(영수증 출력 결정)
Abstract Operation(Method)	-
Exceptional Courses of Events	No를 받으면 Print Statement는 수행하지 않는다.

Activity 2052. Implements Windows



※ Send Interaction Diagram에 필요한 Windows Name이 모두 내포

Name	Click Input Account ID
Responsibilities	계좌 아이디를 입력하기 위한 버튼을 누른다.
Type	GUI
Cross References	N/A
Notes	'Input Account ID'라는 버튼이 화면에 표시된다.
Pre-Conditions	ATM 초기화면
Post-Conditions	Account ID를 입력할 수 있는 창 표시

Name	Input Account ID
Responsibilities	계좌 아이디를 입력한다.
Type	GUI
Cross References	R1.1
Notes	입력한 계좌 숫자는 화면에 표시된다.
Pre-Conditions	'Input Account ID' 버튼을 누른 상태
Post-Conditions	계좌정보 조회 후 Controller에 정보 적재

Name	Click Category Button
Responsibilities	거래할 기능의 버튼을 누른다.
Type	GUI
Cross References	N/A
Notes	각 거래기능에 대한 버튼이 화면에 표시된다.
Pre-Conditions	계좌에 대한 정보가 적재되어 있는 상태
Post-Conditions	Bank에서 거래의 클래스를 생성

Name	Input pw in pw space
Responsibilities	비밀번호를 입력한다.
Type	GUI
Cross References	R1.2
Notes	입력한 비밀번호 숫자는 화면에 표시된다.
Pre-Conditions	거래할 기능의 클래스가 생성되어 있는 상태
Post-Conditions	비밀번호가 맞는지 확인

Name	Input Account ID
Responsibilities	거래할 액수를 입력한다.
Type	GUI
Cross References	R1.3, 1.4
Notes	입력한 거래 액수 숫자는 화면에 표시된다.
Pre-Conditions	비밀번호가 일치되어 있는 상태
Post-Conditions	수수료가 필요한 기능에는 수수료 책정 후 거래 총액확인

Name	Click Statement Answer
Responsibilities	영수증 출력의사 버튼을 누른다..
Type	GUI
Cross References	R3.1
Notes	영수증 출력에 대한 Y, N버튼이 화면에 표시된다.
Pre-Conditions	모든 거래가 끝난 상태
Post-Conditions	입력된 Boolean값에 따라 영수증 출력 처리

Activity 2055. Write Unit Test Code

1. Controller

```
public class Controller {
    Bank bank;
    public static int User_Account;
    public static int Receiver_Account;
    private static int Password;
    private static String Category;
    public static int Input_Amount;
    private static int User_id;
    private static int User_password;
    private static int User_limit;
    private static int User_Totalmoney;
    private static int User_frequency;
    private static int Receiver_id;
    private static int Receiver_password;
    private static int Receiver_limit;
    private static int Receiver_Totalmoney;
    private static int Receiver_frequency;
    public static String Bank;
    Scanner s=new Scanner(System.in);
```

A. Get Account ID

```
    void get_Account() throws IOException
    {
        System.out.println("input Account number");
        this.User_Account=s.nextInt();
        Bank bank=new Bank();
        bank.Find_info(this.User_Account);
        Account account=new Account();
        this.User_password=account.password();
        this.User_Totalmoney=account.remains();
        this.User_limit=account.Limit();
        this.User_frequency=account.frequency();
```

B. Get Receiver Account ID

```
void get_ReceiverAccount() throws IOException
{
    System.out.println("input Reciever Account number");
    this.Receiver_Account=s.nextInt();
    Bank bank=new Bank();
    bank.Find_info(this.Receiver_Account);
    Account raccount=new Account();
    this.Receiver_password=raccount.password();
    this.Receiver_Totalmoney=raccount.remains();
    this.Receiver_limit=raccount.Limit();
    this.Receiver_frequency=raccount.frequency();

}
```

C. Check Password

```
boolean CheckPassword(int Password)
{
    System.out.println(this.User_password);
    if(Password==this.User_password)
    {
        return true;
    }
    else
    {
        System.out.println("password missmatched ");
        return false;
    }
}
```

D. Input Password

```
void input_Password()
{
    boolean check;
    while(true)
    {
    System.out.println("input password");
    this.Password=s.nextInt();
    if((this.Password)>9999)
    {
        System.out.println("input 4 digit plz");
    }

    check=CheckPassword(this.Password);
    if(check==true)
    {
        System.out.println("password checked");
        break;
    }
    else if(check==false)
    {
        System.out.println("try again plz");
    }
    }

}
```

E. Input Amount

```
void input_Amount()
{
    int check=0;
    while(true)
    {
    System.out.println(" input Amount");
    Input_Amount=s.nextInt();
    System.out.println(Input_Amount+"is it correct? press 1 to proceed");
    check=s.nextInt();
    if(check==1)
    {
        break;
    }
    }

}
```

F. Limit Amount

```
boolean Limit_Amount()
{
    if(Input_Amount>User_limit)
    {

        return false;
    }
    else
    {
        return true;
    }
}
```

G. Category

```
void category(String category) throws IOException
{
    if(category.equals("send"))
    {
        boolean limitcheck;
        Send send=new Send();

        System.out.println(this.User_Account);
        System.out.println(this.User_password);
        System.out.println(this.User_Totalmoney);
        System.out.println(this.User_limit);
        System.out.println(this.User_frequency);
        input_Password();
        get_ReceiverAccount();
        while(true)
        {
        input_Amount();
        this.Input_Amount=send.get_Amount(this.Input_Amount);
        System.out.println(this.Input_Amount);
        limitcheck=Limit_Amount();
        if(limitcheck==false)
        {
            System.out.println("your Limit is:"+User_limit+" check the value again");
        }
        else if(limitcheck==true)
        {
            System.out.println("proceed");break;
        }
        }
        int sum=this.User_Totalmoney-this.Input_Amount;
        int sum2=this.User_frequency+1;
        Bank bank=new Bank();
        bank.update_Account(this.User_Account,sum,sum2);
```

```
        int sum=this.User_Totalmoney-this.Input_Amount;
        int sum2=this.User_frequency+1;
        Bank bank=new Bank();
        bank.update_Account(this.User_Account,sum,sum2);
        bank.update_Account(Receiver_Account,Receiver_Totalmoney+Input_Amount,Receiver_frequenc
        Payback payback=new Payback();
        payback.Check_Payback(User_frequency);
        PrintStatement p=new PrintStatement();
        p.set_Amount(sum,this.Input_Amount);
        p.Get_Answer();

    if(category.equals("withdraw"))
    {
        boolean limitcheck;
        Withdraw withdraw=new Withdraw();
        System.out.println(this.User_Account);
        System.out.println(this.User_password);
        System.out.println(this.User_Totalmoney);
        System.out.println(this.User_limit);
        System.out.println(this.User_frequency);
        input_Password();
        while(true)
        {
        input_Amount();
        this.Input_Amount=withdraw.get_Amount(this.Input_Amount);
        limitcheck=Limit_Amount();
        if(limitcheck==false)
        {
            System.out.println("your Limit is:"+User_limit+" check the value again");
        }
        else if(limitcheck==true)
        {
            System.out.println("proceed");break;
        }
        }
        int sum=this.User_Totalmoney-this.Input_Amount;
        int sum2=this.User_frequency+1;
        Bank bank=new Bank();
        bank.update_Account(this.User_Account,sum,sum2);
        Payback payback=new Payback();
        payback.Check_Payback(User_frequency);
        PrintStatement p=new PrintStatement();

           if(category.equals("deposit"))
           {
               Deposit deposit=new Deposit();
               input_Amount();
               deposit.get_Amount(this.Input_Amount);
               this.Input_Amount=deposit.send_Amount();
               bank.update_Account(this.User_Account,this.User_Totalmoney+this.Input_Amount,this.User_frequency+1,U_bank);
               Payback payback=new Payback();
               payback.Check_Payback(User_frequency);
               PrintStatement p=new PrintStatement();
               p.set_Amount(this.User_Totalmoney+this.Input_Amount,this.Input_Amount);
               p.Get_Answer();
           }
           if(category.equals("check remain"))
           {
               CheckRemain checkremain=new CheckRemain();
               checkremain.show_Amount(User_Totalmoney);
               checkremain.Print_Total_Amount();
               Payback payback=new Payback();
               payback.Check_Payback(User_frequency);
               PrintStatement p=new PrintStatement();
               p.set_Amount(this.User_Totalmoney+this.Input_Amount,this.Input_Amount);
               p.Get_Answer();
           }
       }
}
```

2. Send

```
public class Send {
    Commission commission=new Commission();
    public static int Amount;
    public static int Receiver_Account;

    int get_Amount(int amount){
        int result;
        this.Amount=amount;
        result=send_Amount();
        return result;
    }
    int send_Amount(){
        commission.get_Commission(Amount);
        this.Amount=commission.get_TotalAmount();
        return this.Amount;
    }
}
```

3. Withdraw

```
public class Withdraw {
    private static int Amount;

    int get_Amount(int amount){
        int result;
        this.Amount=amount;
        result=send_Amount();
        return result;
    }
    int send_Amount()
    {
        Commission commission=new Commission();
        commission.get_Commission(Amount);
        this.Amount=commission.get_TotalAmount();
        return this.Amount;
    }
}
```

4. Print Statement

```
public class PrintStatement {
    public static int Changed_Amount;
    public static int Exchanged_Amount;

    void Get_Answer()
    {
        String answer=null;
        Scanner s=new Scanner(System.in);
        System.out.println("would yout print the reciept? press Y/N");
        answer=s.nextLine();
        if(answer.equals("Y"))
        {
            Print_Statement(Changed_Amount,Exchanged_Amount);
        }
        else
        {
            System.out.println("proecdure complete");
        }
        s.close();
    }
    void Print_Statement(int changed_Amount, int Exchanged_amount)
    {
        int g_code;
        Payback payback=new Payback();
        g_code=payback.get_Gift_code();
        System.out.println("Changed_Amount="+changed_Amount);
        System.out.println("Exchanged_Amount="+Exchanged_amount);
        System.out.println("Gift_code="+g_code);


    }

  void set_Amount(int changed_Amount, int exchanged_amount)
  {
      this.Changed_Amount=changed_Amount;
      this.Exchanged_Amount=exchanged_amount;
  }
```

5. Account

```
public class Account extends Bank{
    private static int Account;
    public static int Password;
    public static int Total_Amount;
    public static int Limit_Amount;
    public static int Use_frequency;

    public int account(){//load로 활용해서 어떻게 구현할지 생각이 안나서 일일이 그냥 만
        this.Account=info[0];//controller가 메인될지 모르겠지만 거기서 이거 하나씩
        return this.Account;
    }
    public int password(){
        this.Password=info[1];
        return this.Password;
    }
    public int remains(){
        this.Total_Amount=info[2];
        return this.Total_Amount;
    }
    public int Limit(){
        this.Limit_Amount=info[3];
        return this.Limit_Amount;
    }
    public int frequency(){
        this.Use_frequency=info[4];
        return this.Use_frequency;
    }
}
```

6. Bank

```
public class Bank {
    private static File file;
    private static String Bank_name;
    public static int info[] = {0,0,0,0,0};
    void Find_info(int Account) throws IOException
    {
        if(Account<20000&&Account>10000)//계좌 형식으로 이렇게 일단 만들어놓고 하면 될듯
        {
            Bank_name="Shin han.txt";
        }
        else if(Account<30000&&Account>20000)
        {
            Bank_name="kuk min.txt";
        }
        else if(Account<40000&&Account>30000)
        {
            Bank_name="IBK.txt";

        }
        this.file=new File(Bank_name);
        FileReader fReader=new FileReader(file);
        BufferedReader breader=new BufferedReader(fReader);
        String str=null;//파일 정보를 입력받기 위한 변수

        int i=0;


   int i=0;

   while((str=breader.readLine())!=null)
   {
       int Id=Integer.parseInt(str);//파일에서 읽어오는 string을 int로 변환 시키려고 한거
       if(Id==Account)//0=> account number,1=>password,2=>Remains,3=>Limit,4=>Frequency
       {            //파일을 훑으면서 맞는 계좌를 찾으면 그때부터 정보 저장 한줄 한줄
           while(i!=4)
           {

           Id=Integer.parseInt(str);
           info[i]=Id;
           i++;
           str=breader.readLine();
           }
       }
   }
   fReader.close();
   breader.close();
```

```
void update_Account(int Account_id, int Changed_Amount,int Use_frequency) throws IOException
{

    String dummy="";
    BufferedReader br=new BufferedReader(new InputStreamReader(new FileInputStream(file)));
    String line;
    this.info[2]=Changed_Amount;
    this.info[4]=Use_frequency;
    while((line=br.readLine())!=null)
    {
        dummy+=(line+"\r\n");
        int Id=Integer.parseInt(line);
        if(Id==Account_id)
        {
            for(int i=0;i<5;i++)
            {
                String deldata=br.readLine();
            }
        }

    }
    FileWriter fw=new FileWriter(Bank_name);
    for(int i=1;i<5;i++)
    {
        dummy+=(Integer.toString(info[i])+"\r\n");
    }
    fw.write(dummy);
    fw.close();
    br.close();
}
```

7. Payback

A. Check Frequency

```
import java.util.Scanner;
public class Payback {
    private static int Gift_code;

    int check_Frequency(int frequency)
    {

        if(frequency>=10&&frequency<20)
        {
            return 1;
        }
        else if(frequency>=20&&frequency<30)
        {
            return 2;
        }
        else if(frequency>=30&&frequency<40)
        {
            return 3;
        }
        else
        {
            return -1;
        }
    }
```

B. Check Payback

```
void Check_Payback(int frequency)
{
    int option=0;
    if(frequency>=10)
    {
        option=check_Frequency(frequency);
        switch(option)
        {
        case -1:
            System.out.println("payback not available");
        case 1:
            Choose_Gift_code(1);break;
        case 2:
            Choose_Gift_code(2);break;
        case 3:
            Choose_Gift_code(3);break;
        }
    }
}
```

C. Choose Gift Code

```
void Choose_Gift_code(int option)
{
    int choice=0;
    Scanner s= new Scanner(System.in);
    System.out.println("choose one you want to get");
    if(option==1)
    {
        System.out.println("1= cofee 2= beverage");
        choice=s.nextInt();
        switch(choice)
        {
        case 1:
            this.Gift_code=11; break;
        case 2:
            this.Gift_code=12; break;
            default:
                System.out.println("invalid choice choose again");
        }
    }
    else if(option==2)
    {
        System.out.println("1=piece cake 2=piece pizza");
        choice=s.nextInt();
        switch(choice)
        {
        case 1:
            this.Gift_code=21; break;
        case 2:
            this.Gift_code=22; break;
            default:
                System.out.println("invalid choice choose again");
        }
    }
    else if(option==3)
    {
        System.out.println("1=rice packet 2=5000won");
        choice=s.nextInt();
        switch(choice)
        {
        case 1:
            this.Gift_code=31; break;
        case 2:
            this.Gift_code=32; break;
            default:
                System.out.println("invalid choice choose again");
        }
    }
}
```

8. Deposit

```
public class Deposit {
    private static int Amount;

    void get_Amount(int amount)
    {
        this.Amount=amount;
    }

    int send_Amount(int amount)
    {
        return this.Amount;
    }

}
```

9. Check Remain

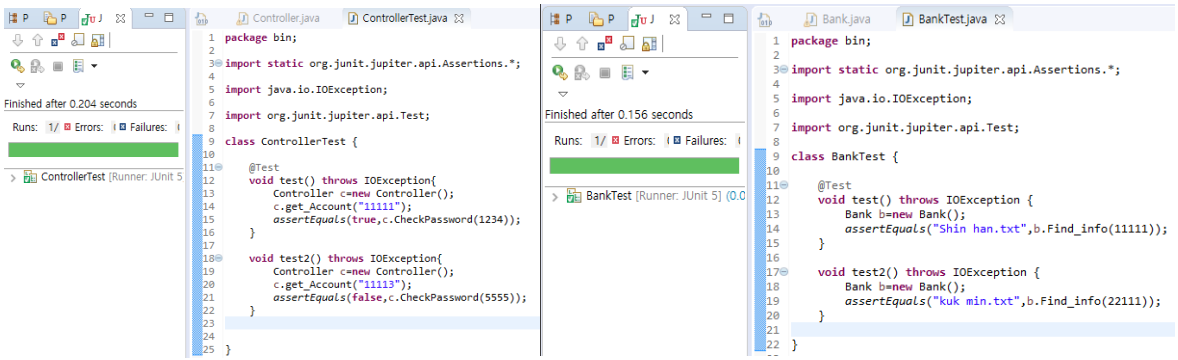
```
public class CheckRemain {
    private static int Remain;
    void show_Amount(int amount)
    {
        Check_TotalAmount(amount);
    }
    void Check_TotalAmount(int amount)
    {
        this.Remain=amount;
    }
    void Print_Total_Amount()
    {
        System.out.println("your remain money id: "+Remain);
    }
}
```

10. Commission

```
import java.text.SimpleDateFormat;
public class Commission {
    private static int Current_Time;
    public static int Commission;
    public static int Total_Amount;

    void get_Commission(int amount){
        check_Currenttime();
        if(Current_Time<22)
        {
            Commission=50;
        }
        else
        {
            Commission=100;
        }
        count_commission(amount);
            }
    void check_Currenttime(){
        Date date=new Date();
        SimpleDateFormat e1=new SimpleDateFormat("H
        String time=e1.format(date);
        Current_Time=Integer.parseInt(time);
    }
    void count_commission(int amount){
        Total_Amount=amount+Commission;
    }
    int get_TotalAmount(){
        return Total_Amount;
    }
}
```

Activity 2061. Unit Testing

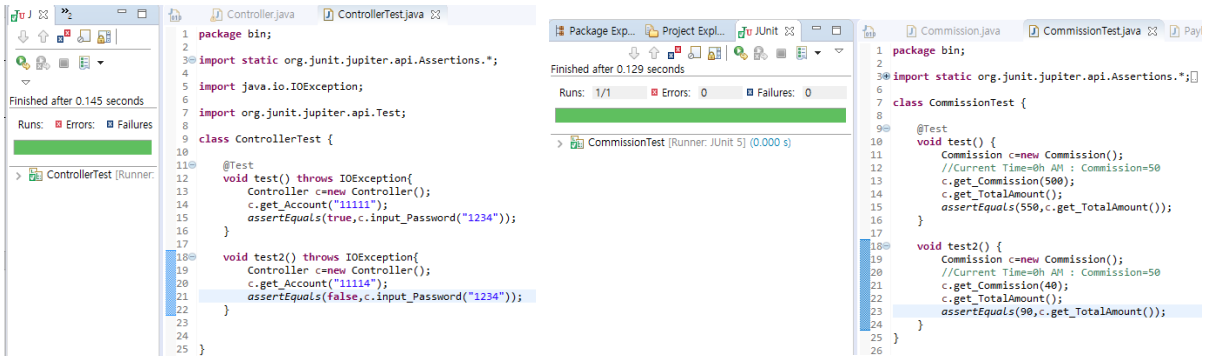
1) Check Password/Find Info



```
package bin;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;

import org.junit.jupiter.api.Test;

class ControllerTest {

    @Test
    void test() throws IOException{
        Controller c=new Controller();
        c.get_Account("11111");
        assertEquals(true,c.CheckPassword(1234));
    }

    void test2() throws IOException{
        Controller c=new Controller();
        c.get_Account("11113");
        assertEquals(false,c.CheckPassword(5555));
    }

}
```

```
package bin;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;

import org.junit.jupiter.api.Test;

class BankTest {

    @Test
    void test() throws IOException {
        Bank b=new Bank();
        assertEquals("Shin han.txt",b.Find_info(11111));
    }

    void test2() throws IOException {
        Bank b=new Bank();
        assertEquals("kuk min.txt",b.Find_info(22111));
    }
}
```

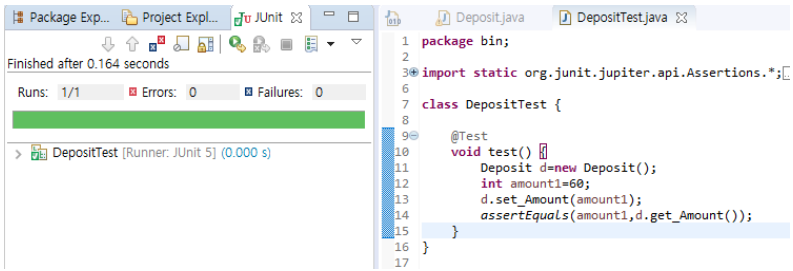
2) Input Password/Get Commission/Get Total Amount



```
package bin;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;

import org.junit.jupiter.api.Test;

class ControllerTest {

    @Test
    void test() throws IOException{
        Controller c=new Controller();
        c.get_Account("11111");
        assertEquals(true,c.input_Password("1234"));
    }

    void test2() throws IOException{
        Controller c=new Controller();
        c.get_Account("11114");
        assertEquals(false,c.input_Password("1234"));
    }

}
```

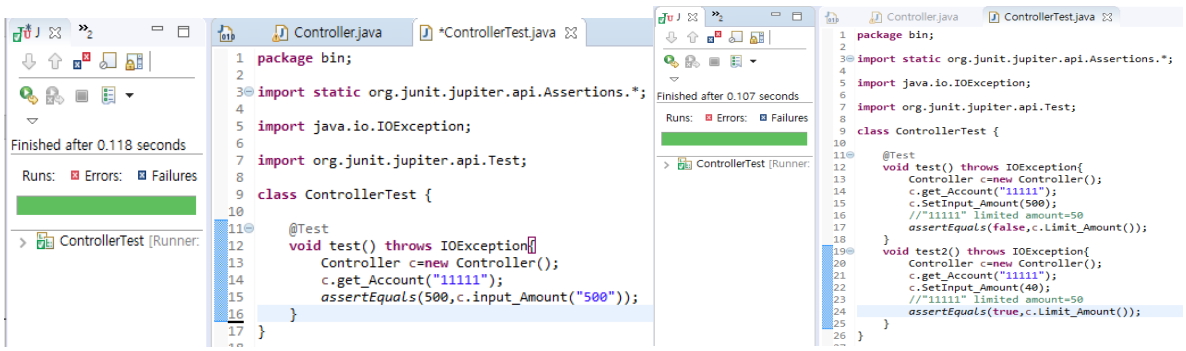
```
package bin;

import static org.junit.jupiter.api.Assertions.*;

class CommissionTest {

    @Test
    void test() {
        Commission c=new Commission();
        //Current Time=0h AM : Commission=50
        c.get_Commission(500);
        c.get_TotalAmount();
        assertEquals(550,c.get_TotalAmount());
    }

    void test2() {
        Commission c=new Commission();
        //Current Time=0h AM : Commission=50
        c.get_Commission(40);
        c.get_TotalAmount();
        assertEquals(90,c.get_TotalAmount());
    }
}
```

3) Get Amount



```
package bin;

import static org.junit.jupiter.api.Assertions.*;

class DepositTest {

    @Test
    void test() {
        Deposit d=new Deposit();
        int amount1=60;
        d.set_Amount(amount1);
        assertEquals(amount1,d.get_Amount());
    }
}
```

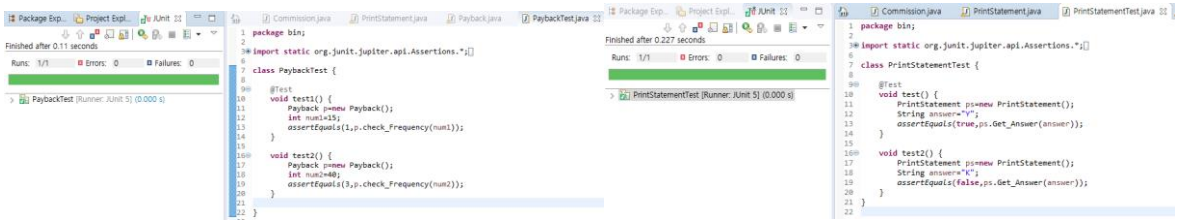
4) Input Amount/Limited Amount



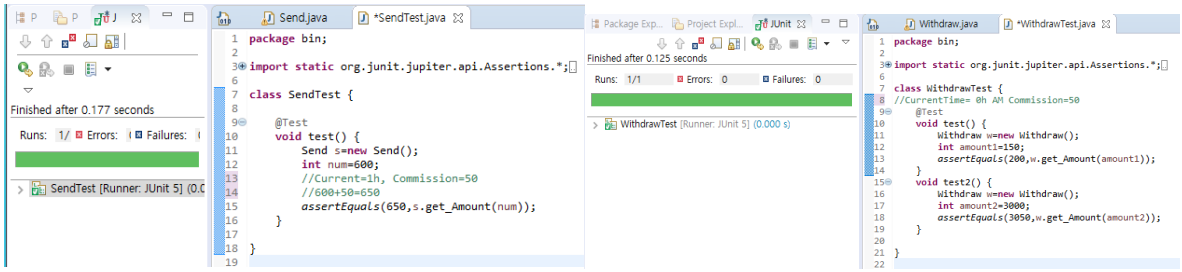
```
package bin;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;

import org.junit.jupiter.api.Test;

class ControllerTest {

    @Test
    void test() throws IOException{
        Controller c=new Controller();
        c.get_Account("11111");
        assertEquals(500,c.input_Amount("500"));
    }
}
```

```
package bin;

import static org.junit.jupiter.api.Assertions.*;

import java.io.IOException;

import org.junit.jupiter.api.Test;

class ControllerTest {

    @Test
    void test() throws IOException{
        Controller c=new Controller();
        c.get_Account("11111");
        c.SetInput_Amount(500);
        //"11111" limited amount=50
        assertEquals(false,c.Limit_Amount());
    }
    void test2() throws IOException{
        Controller c=new Controller();
        c.get_Account("11111");
        c.SetInput_Amount(40);
        //"11111" limited amount=50
        assertEquals(true,c.Limit_Amount());
    }
}
```

5) Check Frequency/ Get Answer



6) Get Amount



Activity 2063. System Testing

Num	Test항목	Test내용	Use Case	Func	P/F
1-1	Input Account ID	계좌번호를 입력한다.	Find Info	R1.1	P
1-2	Load Info	입력된 계좌번호에 해당되는 정보를 찾아 Controller에 적재한다.	Find Info	R1.1	P
2-1	Input Password	비밀번호를 입력한다.	Check Password	R1.2	P
2-2	Check Password	비밀번호가 맞는지 확인한다.	Check Password	R1.2	P
3-1	Get Amount	입력 거래액수를 받는다.	Count Commission	R1.4	P
3-2	Get Total(total_amount)	책정된 수수료와 입력 받은 거래액수를 더하여 각 거래 클래스에 넘겨준다.	Count Commission	R1.4	P
3-3	Get Commission	수수료를 계산하기 전 송/출금액을 받아온다.	Count Commission	R1.4	P
3-3	Limited Amount	거래해야 할 액수가 한도내에 있는지 확인한다.	Limited Amount	R1.3	P
4-1	Get Answer	영수증 출력의사를 입력한다.	Print Statement	R3.1	P

4-2	Print Statement	입력된 의사에 따라 영수증 출력기능을 수행한다.	Print Statement	R3.1	P
5-1	Check Payback	거래횟수를 조회하여 환급대상인지 판별한다.	Payback	R3.2	P
5-2	Check Frequency	계좌의 사용횟수를 조회한다.	Payback	R3.2	P

Activity 2066. Testing Traceability Analysis