CS4001NI Programming

30% Individual Coursework

2022-23 Autumn

Student Name: Rounak Pradhan

London Met ID: 22066975

College ID: NP01NT4A220198

Group: L1N6

Assignment Due Date: Friday, January 27, 2023

Assignment Submission Date: Friday, January 27, 2023

Table of Contents

Table of figure

1.Introduction

1.1. Introduction of project

The object-oriented Java concept is used int this project which is based on real world software scenario and involves designing a class to represent a Bank card. To represent a debit card and a credit card, respectively, along with its two subclasses. In this case, three classes Bank card, Debit card, and Credit card are created with Bank card serving as the superclass of the other two. Class, Object, Encapsulation, and Inheritance are the four principles of Object oriented programming that are applied here.

In three classes Bank card, Debit card, and credit card each class attributes are written in private access modifier. We only use getter and setter method to access attributes which directly denotes encapsulation principles. Super class and sub classes are example of Inheritance principle which also used.

1.2. Tools used in project

To complete this project many tools are used some of them are listed below:

- Blue j
- Ms-word
- Draw.io

Blue j

A Blue j java program is one that was created using the Blue j development environment and written in the java programming language. The program can be made by starting a new project in Blue j and then adding oen or more classes to it. These classes can contain methods and fields that define the programs behaviour and data. After the classes have been defined, they can be instantiated and their methods called to carry out the program's functionally. Blue j also offers a variety of debugging and testing capabilities, including the capacity to walk through the code, set breakpoints, and examine the values of variables. Additionally, Blue j enables you to interact with the program by constructing and manipulating objects belonging to the established classes, which can aid in your comprehension of how the program functions and the detection of any mistakes.

Figure 1 of Blue j

Ms-word

Microsoft Word is a word processing program that is frequently referred to as just Word. It is utilized for documentation creation, editing, and formatting and is a component of the Microsoft Office product line. Users using

word can generate documents with professional appearances, including letters, resumes, reports and more.

Figure 2 of Ms-word

Draw.io

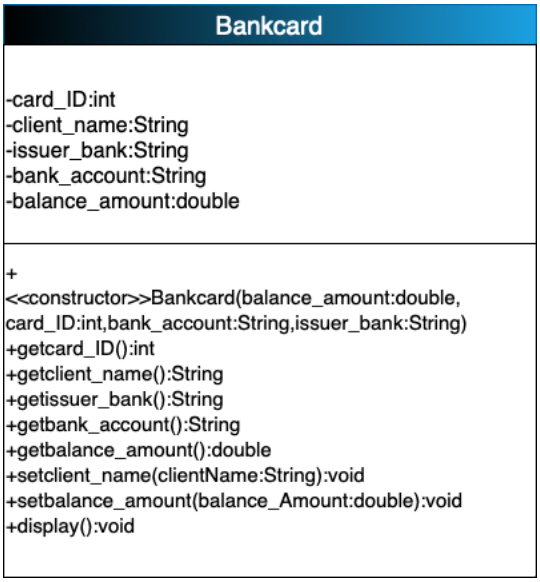
Users can create and edit diagrams, flowcharts, network diagrams, and other sorts of visual representations using the web-based diagramming tool draw.io. It is a free and open-source application for creating and sharing diagrams online, and it works with a variety of file formats, including Visio, Gliffy, and Lucidchart. Users can work together in real-time on diagrams, share them via links, or export the as images, PDFs, or other file types.

Figure 3 of draw.io

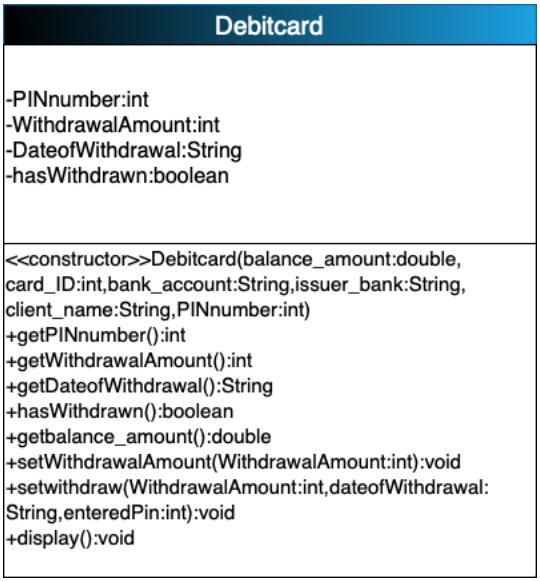
2.Class Diagram

In software engineering and object-oriented design, class diagrams are a sort of diagram that are frequently used to describe a syastem's structure by outlining its classes, interfaces, and constructor between them.

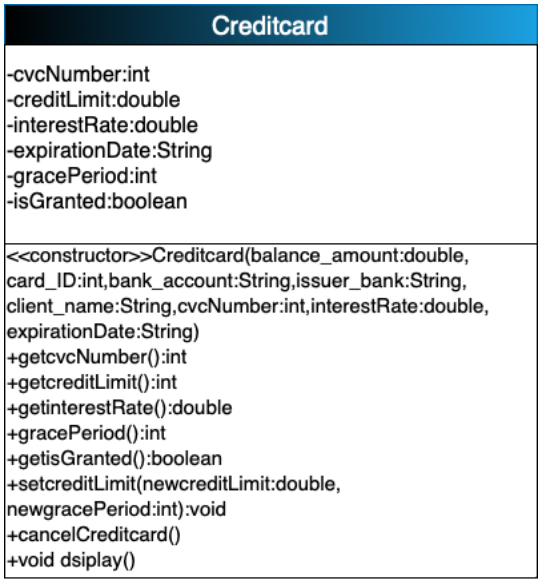
2.1. Bankcard



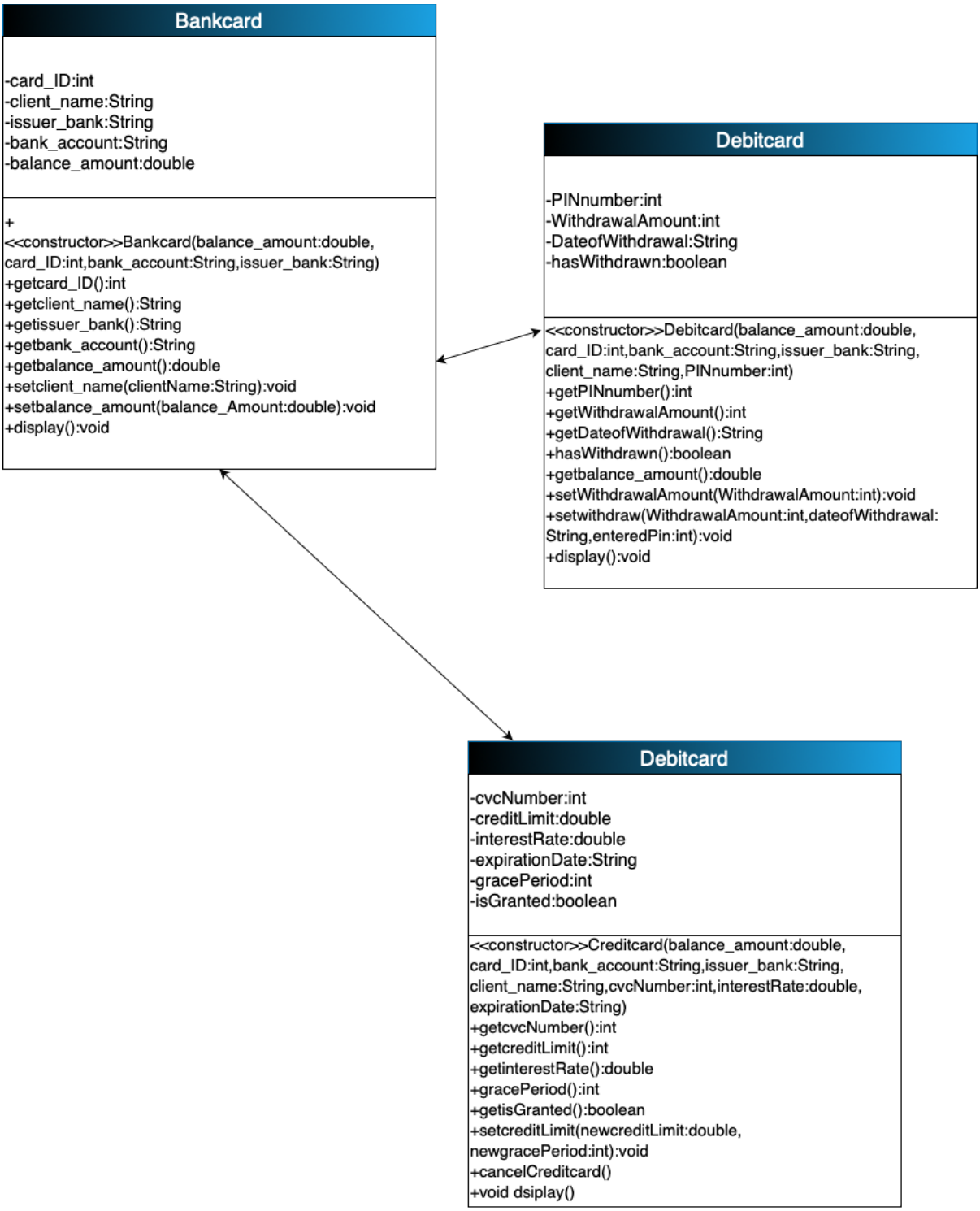
2.2. Debitcard



2.3.Credit card



Relationship between classes


Bankcard
-card_ID:int
-client_name:String
-issuer_bank:String
-bank_account:String
-balance_amount:double
+
<<constructor>>Bankcard(balance_amount:double,
card_ID:int,bank_account:String,issuer_bank:String)
+getcard_ID():int
+getclient_name():String
+getissuer_bank():String
+getbank_account():String
+getbalance_amount():double
+setclient_name(clientName:String):void
+setbalance_amount(balance_Amount:double):void
+display():void
Debitcard
-PINnumber:int
-WithdrawalAmount:int
-DateofWithdrawal:String
-hasWithdrawn:boolean
<<constructor>>Debitcard(balance_amount:double,
card_ID:int,bank_account:String,issuer_bank:String,
client_name:String,PINnumber:int)
+getPINnumber():int
+getWithdrawalAmount():int
+getDateofWithdrawal():String
+hasWithdrawn():boolean
+getbalance_amount():double
+setWithdrawalAmount(WithdrawalAmount:int):void
+setwithdraw(WithdrawalAmount:int,dateofWithdrawal:
String,enteredPin:int):void
+display():void
Debitcard
-cvcNumber:int
-creditLimit:double
-interestRate:double
-expirationDate:String
-gracePeriod:int
-isGranted:boolean
<<constructor>>Creditcard(balance_amount:double,
card_ID:int,bank_account:String,issuer_bank:String,
client_name:String,cvcNumber:int,interestRate:double,
expirationDate:String)
+getcvcNumber():int
+getcreditLimit():int
+getinterestRate():double
+gracePeriod():int
+getisGranted():boolean
+setcreditLimit(newcreditLimit:double,
newgracePeriod:int):void
+cancelCreditcard()
+void dsiplay()

3. Pseudocode

A type of informal high-level depiction of the logic of an algorithm or computer program is known as pseudocode. The logic of a program is described using a combination of normal language and programming language elements, but it is not executable code. Before any actual coding is done, pseudocode is frequently used to help plan and arrange the program's logic and structure during the early phases of development. It can also be used to describe and document the logic of already written programs.

3.1. Pseudocode for Bankcard class

CREATE a class Bankcard as public

DO

DECLARE an instance variable card_ID of int data as private

DECLARE an instance variable client_name of String data as private

DECLARE an instance variable issuer_bank as String data as private

DECLARE an instance variable bank_account as String data as private

DECLARE an instance variable balance_amount as double data as private

CREATE a constructor which accepts five parameters double balance_amount int card_ID String bank_acount String issuer_bank

DO

INITIALIZE the value to instance variable card_ID to card_ID

INITIALIZE the value of instance variable client_name to empty String

INITIALIZE the value of instance variable bank_account to bank_account

INITIALIZE the value of instance variable balance_amount to balance_amount

END DO

CREATE an instance getters method getcard_ID to return type int

DO

RETURN the value of instance variable card_ID

END DO

CREATE an instance getter method getclient_name to return type String

DO

RETURN the value of instance variable client_name

END DO
CREATE an instance getter method getissuer_bank to return type String

DO

RETURN the value of instance variable issuer_bank

END DO

CREATE an instance gettter method getbank_account to return type String

DO

RETRUN the value of instance variable bank_account

END DO

CREATE an instance getter method getbalance_amount to return type double

DO

RETURN the value of instance variable balance_amount

END DO

CREATE an instance setter methos setclient_name which accepts parameter String clientName

DO

INITIALIZE the value of instance variable client_name to clientName

END DO

CREATE an instance setter method setbalance_amount which accepts parameter double balance_Name

DO

INITIALIZE the value of instance variable balance_amount to balance_Name

END DO

CREATE an instance method to display

DO
DISPLAY card_ID
IF client_name is empty

DISPLAY “Client name: Not assigned”
ELSE

DISPLAY “Client name”
END IF

DISPLAY issuer_bank
DISPLAY bank_account
DISPLAY balance_amount
END DO
END DO

3.2. Pseudocode for Debitcard

CREATE a child class Debitcard which extends parent class Bankcard

DO

DECLARE an instance variable PINnumber of int data type as private

DECLARE an instance variable WithdrawalAmount of int data type as private

DECLARE an instance variable DateofWithdrawal of String data type as private

DECLARE an instance variable hasWithdrawn of Boolean data type as private

CREATE a constructor which accepts six parameters int Balance_amount int card_ID String bank_account String Issuer_bank String client_name and int PINnumber

DO

CALL superclass constructor with four parameters

CALL setter method of client_name with parameter from superclass

INITIALIZE the value of instance variable PINnumber to PINnumber

SET instance variable hasWithdrawn to false

END DO

CREATE an instance getter method getPINnumber of return type int

DO

RETURN the value of instance variable PINnumber

END DO

CREATE an instance getter method getWithdrawalAmount of return type int

DO

RETURN the value of instance variable WithdrawalAmount

END DO

CREATE an instance getter method getDateofWithdrawal of return type String

DO

RETURN the value of instance variable DateofWithdrawal

END DO

CREATE an instance getter method gethasWithdrawn of return type Boolean

DO

RETURN the value of instance variable hasWithdrawn

END DO

CRAETE an instance setter method setWithdrawlAmount which accepts parameter int WithdrawalAmount

DO

INITIALIZE the value of instance variable WithdrawalAmount to WithdrawalAmount

END DO

CREATE an instance method withdraw which accepts parameters int withdrawalAmount String dateofWithdrawal int enteredPin

DO

IF instance variable enteredPin equals to Pinnumber

IF WithdrawalAmount is less than or equal to Balance_amount of super class

INITIALIZE the value of instance variable WithdrawalAmount to withdrawalAmount

INITIALIZE the value of instance variable DateofWithdrawal

UPDATE hasWithdrawn equals to true

ELSE

DISPLAY “Insufficient Balance”

END IF

3.3. Pseudocode for Creditcard

CREATE a child Debitcard which extends parent class Bankcard

DO

DECLARE an instance variable cvcnumber of int data type as private

DECLARE an instance variable creditLimit of double data type as private

DECLARE an instance variable interestRate of double data type as private

DECLARE an instance variable expirationDate of String data type as private

DECLARE an instance variable gracePeriod of int data type as private

DECLARE an instance variable isGranted of Boolean data type as private

CREATE a constructor which accepts eight parameters double Balance_amount int card_ID String bank_account String Issuer_bank String client_name int cvcnumber double interestRate String expirationDate

DO

CALL superclass constructor with four parameters

CALL setter method of client_name with its parameter from superclass

INITIALIZE the value of instance variable cvcNumber to cvcNumber

INITIALIZE the value of instance variable interestRate to interestRate

SET the value of instance variable isGranted to false

END DO

CREATE an instance gettter method getcvcNumber of return type int

DO

RETURN the value of instance variable cvcNumber

END DO

CREATE an instance getter method getcreditLimit of return type double

DO

RETURN the value of instance variable creditLimit

END DO

CREATE an instance getter method getinterestRate of return type double

DO

RETURN the value of instance variable interestRate

END DO

CREATE an instance getter method getgracePeriod of return type int

DO

RETURN the vaue of instance variable gracePeriod

END DO

CREATE an instance getter method getisGranted of return type Boolean

DO

RETURN the value of instance variable isGranted

END DO

CREATE a setter method setcreditLimit which accepts two parameters: double newcreditLimit int newgracePeriod

DO

IF newcreditLimit is less than or equal to 2.5 times Balance_amount of superclass

    INITIALIZE the value of instance variable creditLimit to newcreditLimit

    INITIALIZE the value of instance variable gracePeriod to newgracePeriod

    UPDATE the value of isGranted to true

END IF

IF isGranted is false

    IF creditLimit is greater than 2.5 times Balance_amount of super class

    DISPLAY “Credit cannot be issued”

END IF

END IF

END DO

CREATE an instance void method cancecreditcard

DO

INITIALIZE the value of instance variable cvcNumber to 0

INITIALIZE the value of instance variable creditLimit to 0

INITIALIZE the value of instance variable gracePeriod to 0

UPDATE the value of instance variable isGranted to false

END DO’

CREATE an instance void method display()

DO

CALL display method from superclass

DISPLAY value of instance variable cvcNumber

DISPLAY value of instance variable interestRate

DISPLAY value of instance variable expirationDate

IF the value isGranted

DISPLAY value of instance variable creditLimit

DISPLAY value of instance variable gracePeriod

END IF

END DO

4. Method description

Java class behaviour is referred to as methods. Within the class, methods are constructed. By using methods, we may reuse the code without constantly retyping. Java methods may include a single constantly retyping. Java methods may include a single statement or a collection of statements. We can supply paramteres also known as values or variables to this methods. The following terms should be used in the correct syntax when implementing methods: access modifier, method type, return type, method name, and parameters.

Methods which are created and used in the program are listed below:

Public Bankcard (parameters)

This particular kind of procedure is known as a constructor in the Bankcard class. When an object is created, a constructor is automatically executed. This constructor currently accepts four parameters in the corresponding data types of balance_amount, card_ID, bank_account, and issuer_bank.

Public int getcard_ID()

In the Bankcard class this is the getter method for the card_ID. A getter method is used to return an attributes's value; in this instance, thje card_ID attribute value is returned when the method is called. It uses the public access modifier for the int return type.

Public String getclient_name()

This is the Bankcard class getter function for the client_name attribute. The attribute client_name value is returned when this method is executed or invoked. Its return type is String and its access modifier is public.

Public String getissuer_bank()

This is the Bankcard class getter function for the issuer_bank attribute. The attribute issuer_bank value is returned when this method is called. Its return type is String and its access modifier is public.

Public String getbank_account()

This is the Bankcard class getter function for the bank_account attribute. The attribute bank_account value is returned when this method is called. Its return type isString and its access modifier is public.

Public int getbalance_amount()

This is the Bankcard class getter function for the balance_amount attribute. The attribute balance_amount value is returned when this method is called. Its return type is double and its access modifier is public.

Public int setclient_name(parameter)

The setter function for client_name in the Bankcard class is this. The client_name input value is obtained from the user and assigned to an instance variable via the setter method. Here, a value is obtained from the user and supplied to the instance private variable client_name as a parameter. It is written as having no return type, or void.

Public int setbalance_amount(parameter)

The setter function for balance_amount in the Bankcard class is this. Here, a value is obtained from the user and supplied to the instance private variable balance_amount. It is written as having no return type, or void.

Public void display()

Thus, the statements with variables and values are displayed or printed according to the circumstances and our needs. The name and signature of this method are the same across all three classes. The show method has a number of these statements. It has the public access modifier and is type void.

Public Debitcard(parameters)

This particular kind of procedure is known as a constructor in the Bankcard class. When an object is created, a constructor is automatically called. This constructor accepts six inputs in their respective data types: balance_amount, card_ID, bank_account, issuer_bank, client_name and PINnumber.

Public int getPINnumber()

This is the Debitcard class getter function for PINnumber. The PINnumber attribute value is returned in this scenario when this method is called since the getter method is used to return attribute values. It has a public access modifier on its int return type.

Public int getWithdrawalAmount()

This is the Debitcard class getter function fot WithdrawalAmount. The WothdrawalAmount attribute value is returned in this scenario when this method is called since the getter method is used to return attribute values. It has public access modifier on its int return type.

Public String getDateofWithdrawal()

This is the Debitcard class getter function for DateofWithdrawal. The DAteofWithdrawal attribute value is returned in this scenario when this method is called sinc ethe getter method is used to return attribute values. It has public access modifier on its String return type.

Public void withdraw(parameters)

The money is directly taken out of the client account using this way.WithdrawalAmount, DateofWithdrawal and PINnumber are all acceptable paramteres for this procedure in their repective data types. The attribute hasWithdrawn is set to true if a valid PINnumber is entered and there is enough money to withdraw only that amount. The characteristics: balance_amount is determined following Withdrawal.

Public Creditcard(parameters)

This particular class of procedure is known as a constructor in Creditclass. When an object is created, a constructor is automatically called. This constructor currently accepts eight parameters in their respective data types: card_ID, client_name, issuer_bank, balance_amount, cvcNumber, interestRate, and expirationDate. The values of the paramteres balance_amount , card_ID and issuer_bank are assigned to the super class constructor parameters inside of this constructor.

Public int getcvcNumber()

This is the Creditclass getter function for cvcNumber. When this method is called, the value of attribute cvcNumber is returned> getter methods are used to return attribute values. Its return type is an int with the public access modifier.

Public double getcreditLimit(0

This is the Creditclass getter function for creditLimit. When this method is called, the value of the attribute creditLImit is returned. Getter methods are used to return attributes values. Its return type is double with the public access modifier.

Public double getinterestRate()

This is the Creditclass getter function for interestRate. When this method is called, the value of the attribute interstRate is returned. Getter methods are used to return attribute values. Its return type is double with the public access modifier.

Public String getexpirationDate()

This is the Creditclass getter function for expirationDate. When this method is called, the value of the attribute expirationDate is returned. Getter methods are used to return attribute values. Its return type is String with the public access modifier.

Public int gracePeriod()

This is the Creditclass getter function for gracePeriod. When this method is called, the value of the attribute gracePeriod is returned. Getter methods are used to return attribute values. Its return type is itn with the public access modifier.

Public Boolean getisGranted()

This is the Creditclass getter function for isGRanted. When this method is called, the value of the attribute isGranted is returned. Getter methods are used to return attribute values. Its return type is Boolean with public access modifier.

Public void setcreditLimit()

This creditLimit is set using this way. This method accepts input from the user for the parameters creditLimit and gracePeriod in their respective data types. This method has a number of statements with if conditions. It is written as void since there is no return type.

Public void cancelcreditcard()

This method is to cancel the all credit details. Instance variables cvcNumber, creditlImit, and gracePeiod have their value changed to zero when this method is called, and isGranted has its values updated to false.

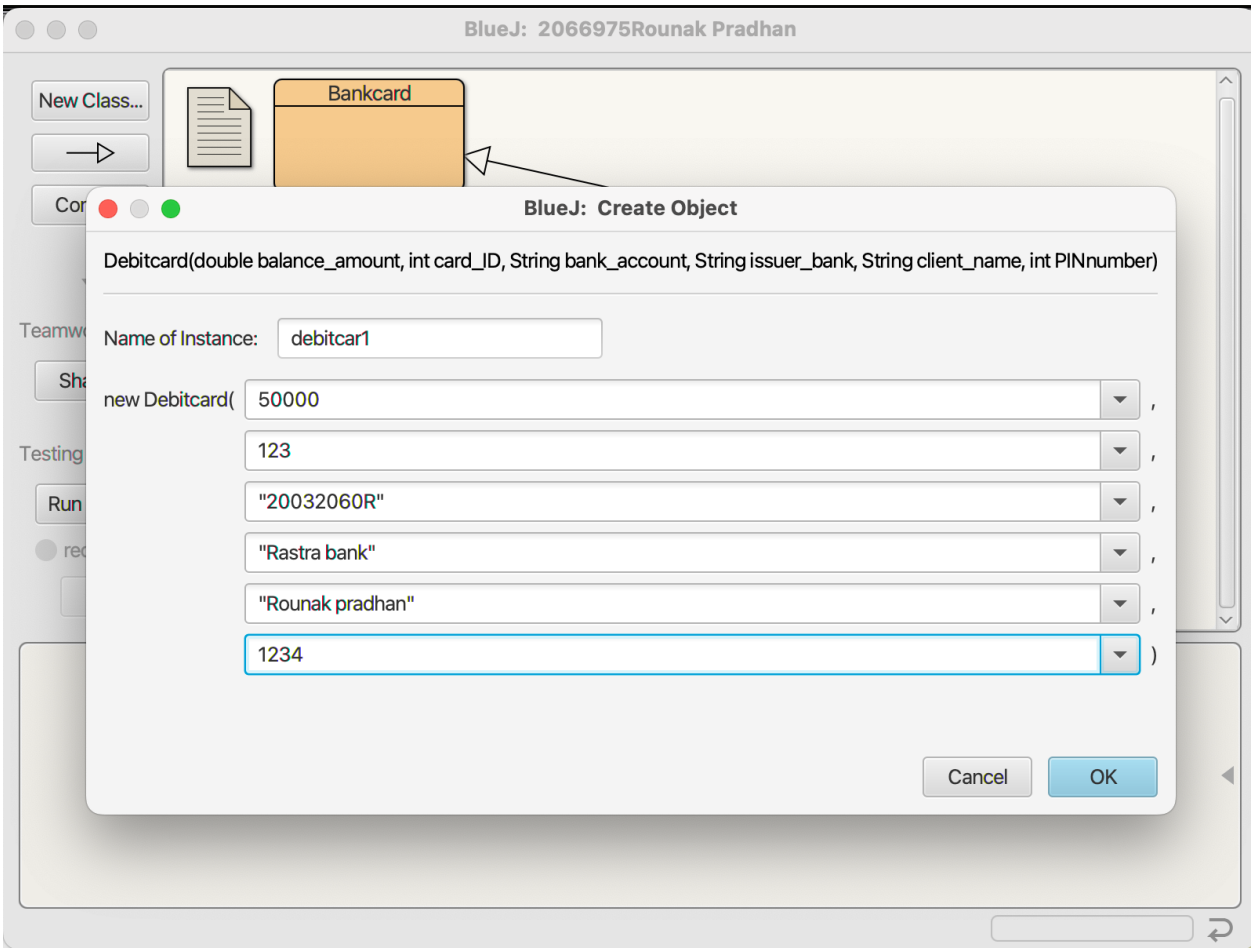
5.Testing

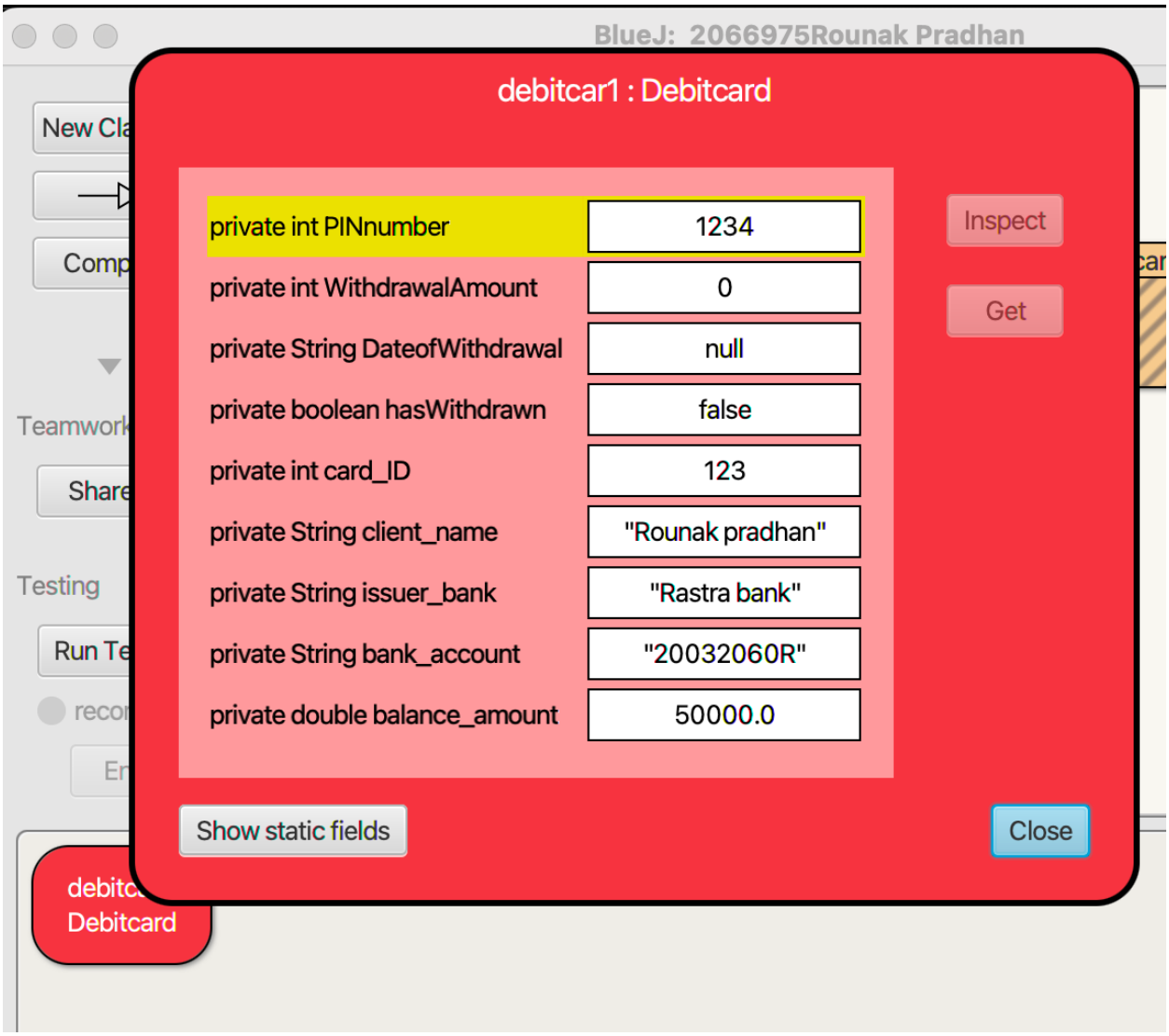
Some testing and screenshots

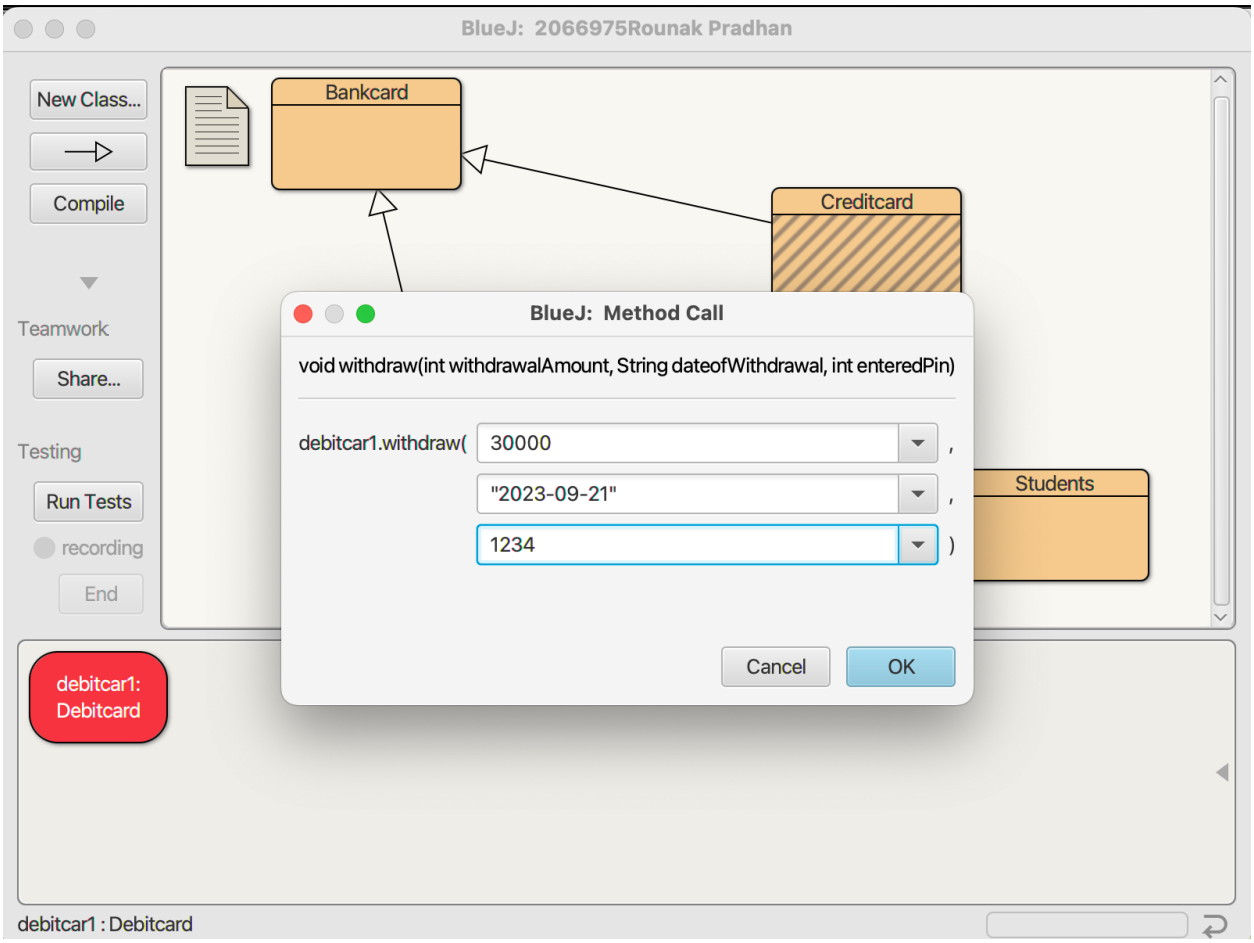
5.1.Test1

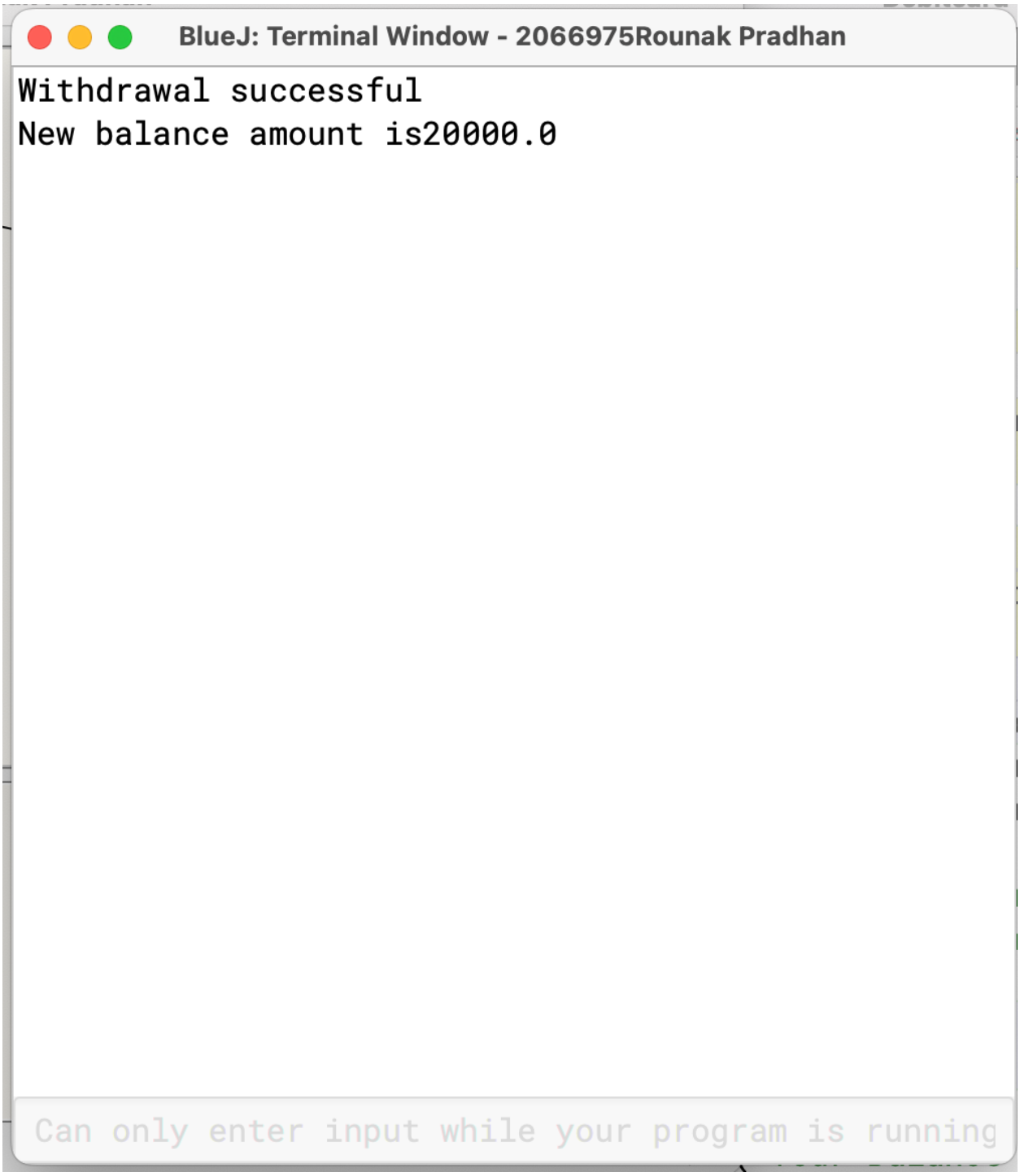
Test number	1
Objective	To inspect the Debitcard class, withdraw the amount, and re-inspect the Debitcard class
Actions	Debitcard is called with the enlisted arguments: Balance_amount:50000 Card_id:123 Bank_account:”20032060R” Issuer_bank:”Rastra bank” client_name:”Rounak Pradhan” PINnumber:1234 Inspection of Debitcard void withdraw was called WithdrawalAmount:30000 DateofWithdrawal:”2023-09-21” enteredPin:1234
Expected result	The amount would be withdrawn
Actual result	The amount was withdrawn
Conclusion	The test was successful.

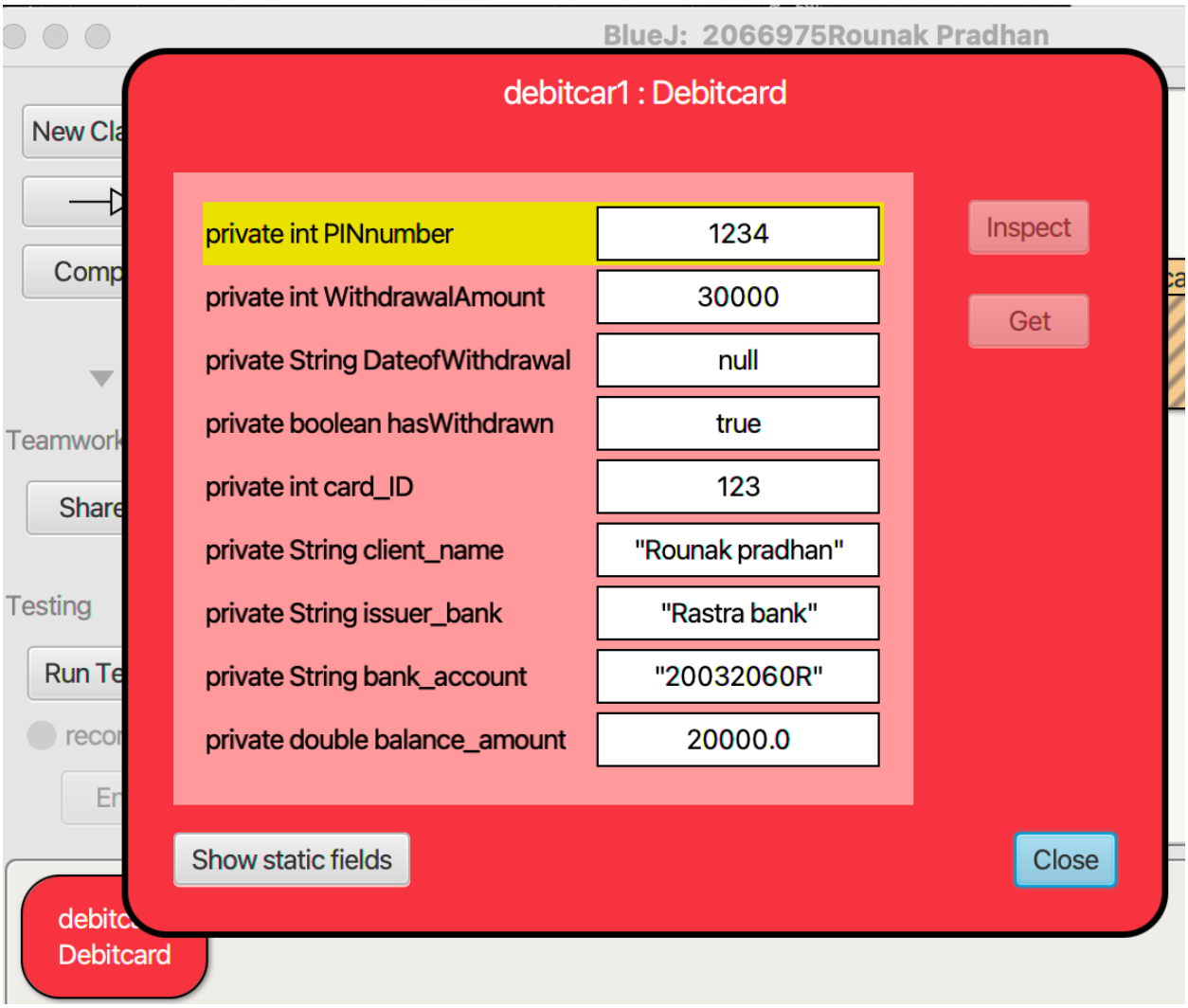
Table 1 Test1


BlueJ: 2066975Rounak Pradhan
New Class...
Bankcard
BlueJ: Create Object
Debitcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int PINnumber)
Name of Instance:
debitcar1
new Debitcard(
50000
123
"20032060R"
"Rastra bank"
"Rounak pradhan"
1234
Cancel
OK


BlueJ: 2066975Rounak Pradhan
debitcar1 : Debitcard
private int PINnumber
1234
private int WithdrawalAmount
0
private String DateofWithdrawal
null
private boolean hasWithdrawn
false
private int card_ID
123
private String client_name
"Rounak pradhan"
private String issuer_bank
"Rastra bank"
private String bank_account
"20032060R"
private double balance_amount
50000.0
Inspect
Get
Show static fields
Close
Debitcard


BlueJ: 2066975Rounak Pradhan
New Class...
Compile
Teamwork
Share...
Testing
Run Tests
recording
End
Bankcard
Creditcard
Students
BlueJ: Method Call
void withdraw(int withdrawalAmount, String dateofWithdrawal, int enteredPin)
debitcar1.withdraw(
30000
"2023-09-21"
1234
Cancel
OK
debitcar1: Debitcard
debitcar1 : Debitcard

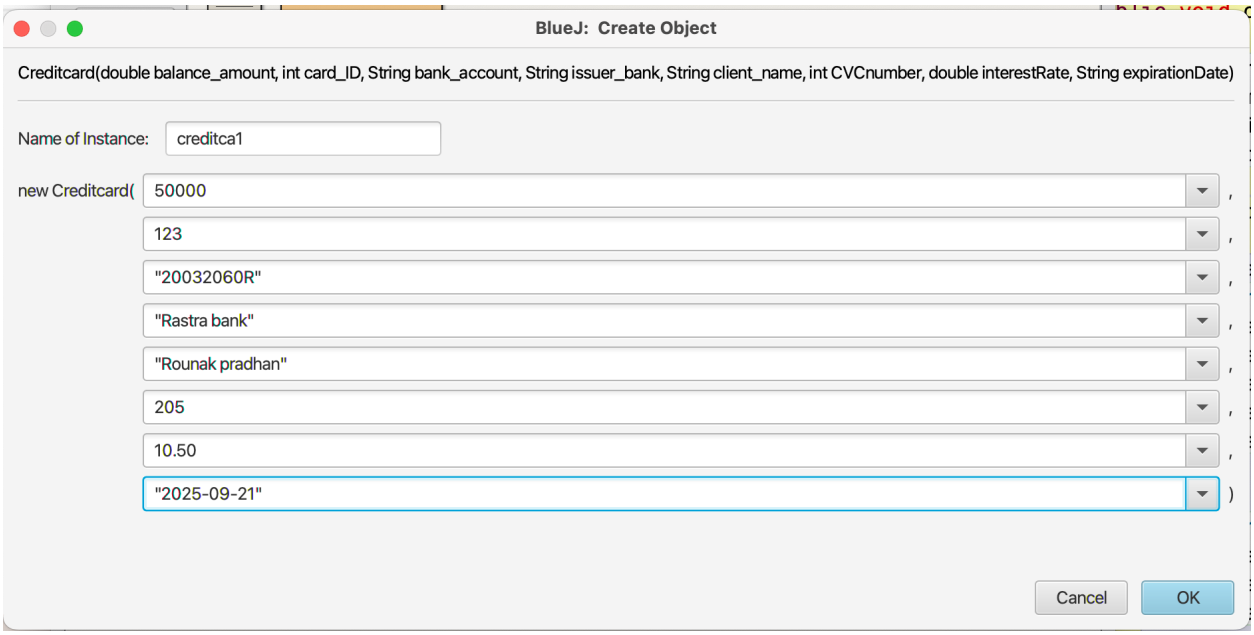

BlueJ: Terminal Window - 2066975Rounak Pradhan
Withdrawal successful
New balance amount is20000.0
Can only enter input while your program is running

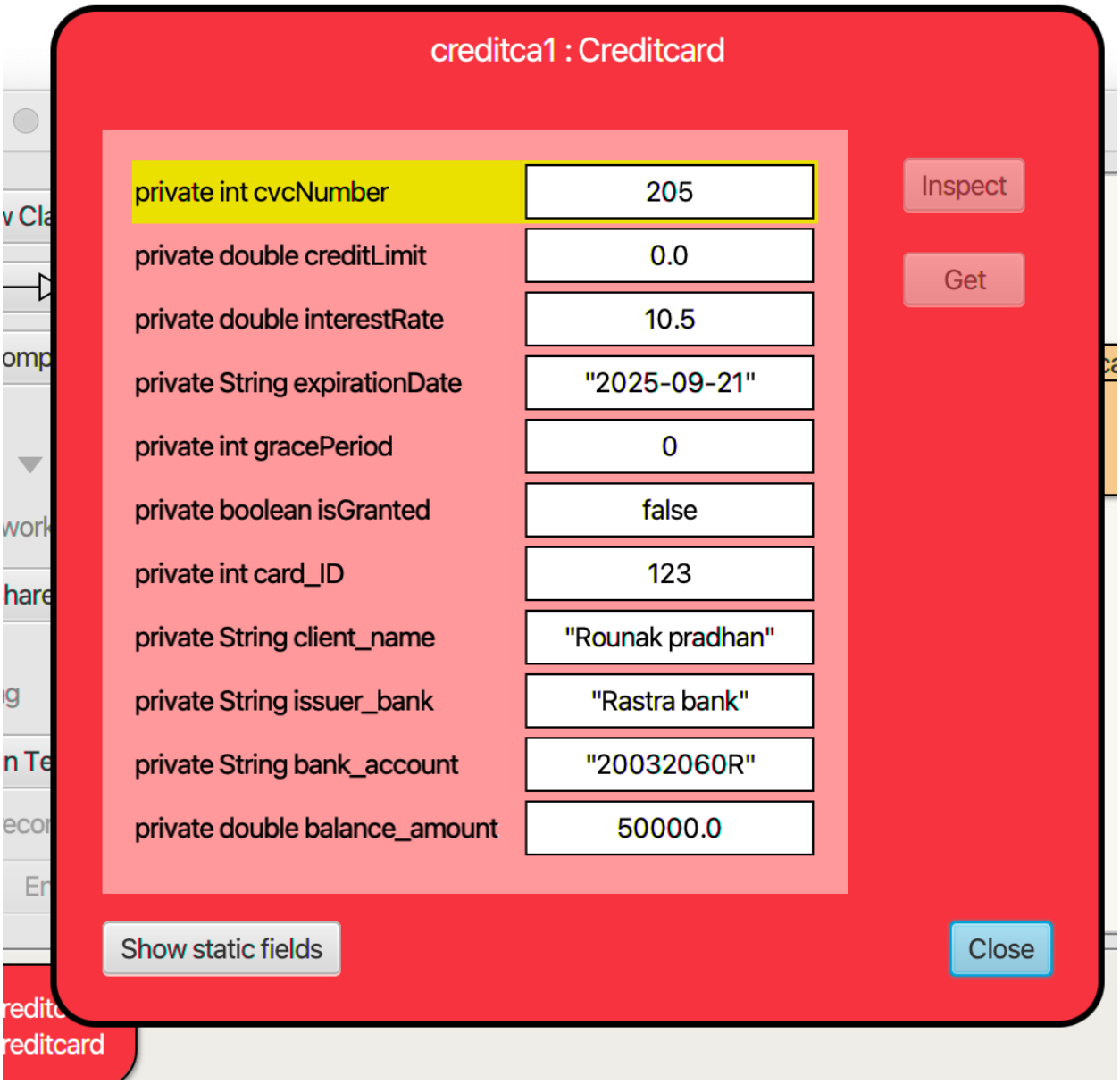

BlueJ: 2066975Rounak Pradhan
debitcar1 : Debitcard
private int PINnumber
1234
private int WithdrawalAmount
30000
private String DateofWithdrawal
null
private boolean hasWithdrawn
true
private int card_ID
123
private String client_name
"Rounak pradhan"
private String issuer_bank
"Rastra bank"
private String bank_account
"20032060R"
private double balance_amount
20000.0
Inspect
Get
Show static fields
Close
Debitcard

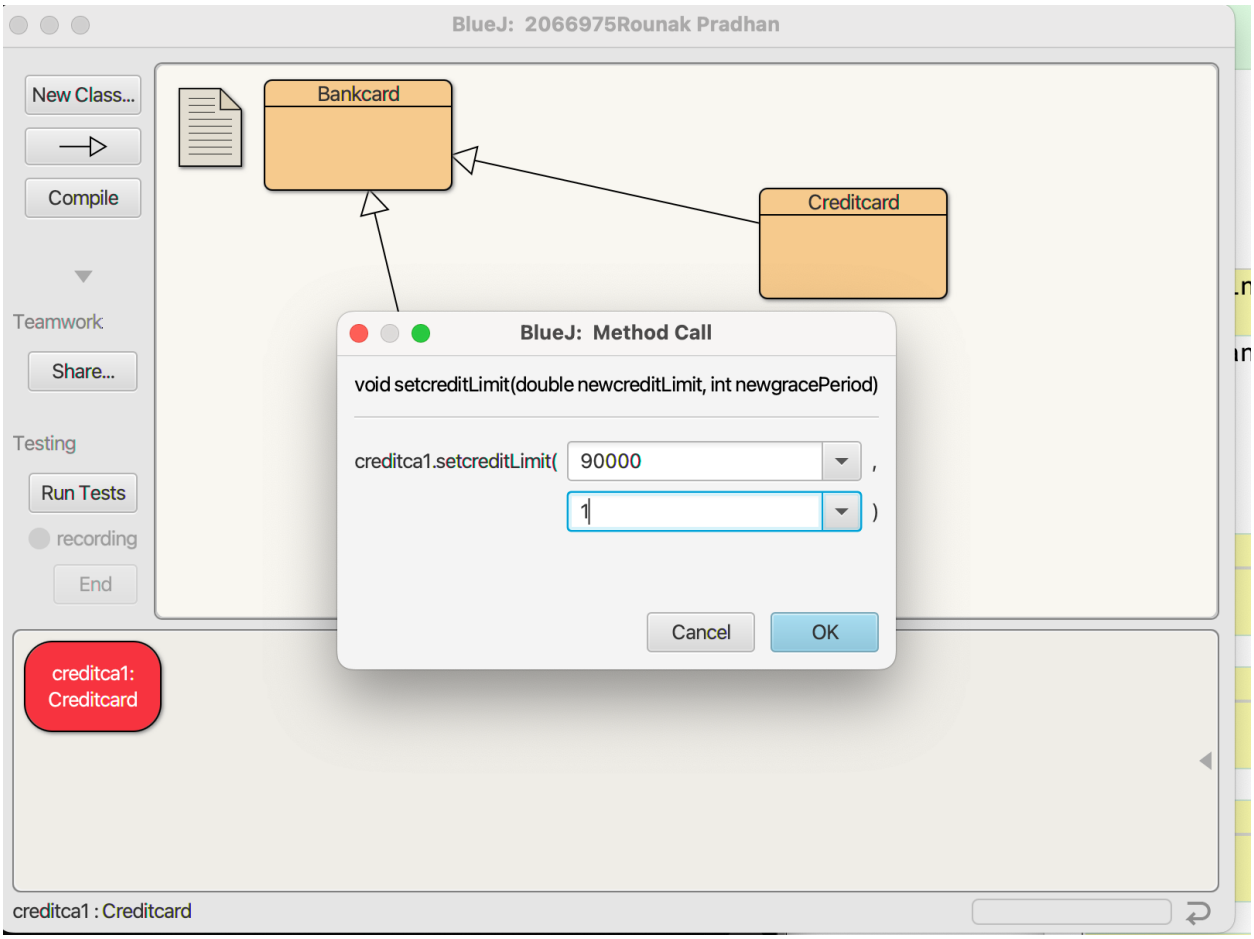
5.2. Test 2

Test number	2
Objective	To inspect the Creditcard class, set the credit limit and reinspect the Credit Card class
Actions	Creditcard is called with the enlisted arguments: Balance_amount:50000 Card_id:123 Bank_account:”20032060R” Issuer_bank:”Rastra bank” client_name:”Rounak Pradhan” cvcNumber:205 interestRate:10.5 expirationDate:”2025-09-21” Inspection of Creditcard void setcreditLimit was called with listed arguments creditLimit:90000 graceperiod:1
Expected result	Credit limit would be set
Actual result	Credit limit was set
Conclusion	The test was successful.

Table 2 Test2


BlueJ: Create Object
Creditcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int CVCnumber, double interestRate, String expirationDate)
Name of Instance:
creditca1
new Creditcard(
50000
123
"20032060R"
"Rastra bank"
"Rounak pradhan"
205
10.50
"2025-09-21"
Cancel
OK


creditca1 : Creditcard
private int cvcNumber
205
private double creditLimit
0.0
private double interestRate
10.5
private String expirationDate
"2025-09-21"
private int gracePeriod
0
private boolean isGranted
false
private int card_ID
123
private String client_name
"Rounak pradhan"
private String issuer_bank
"Rastra bank"
private String bank_account
"20032060R"
private double balance_amount
50000.0
Inspect
Get
Show static fields
Close

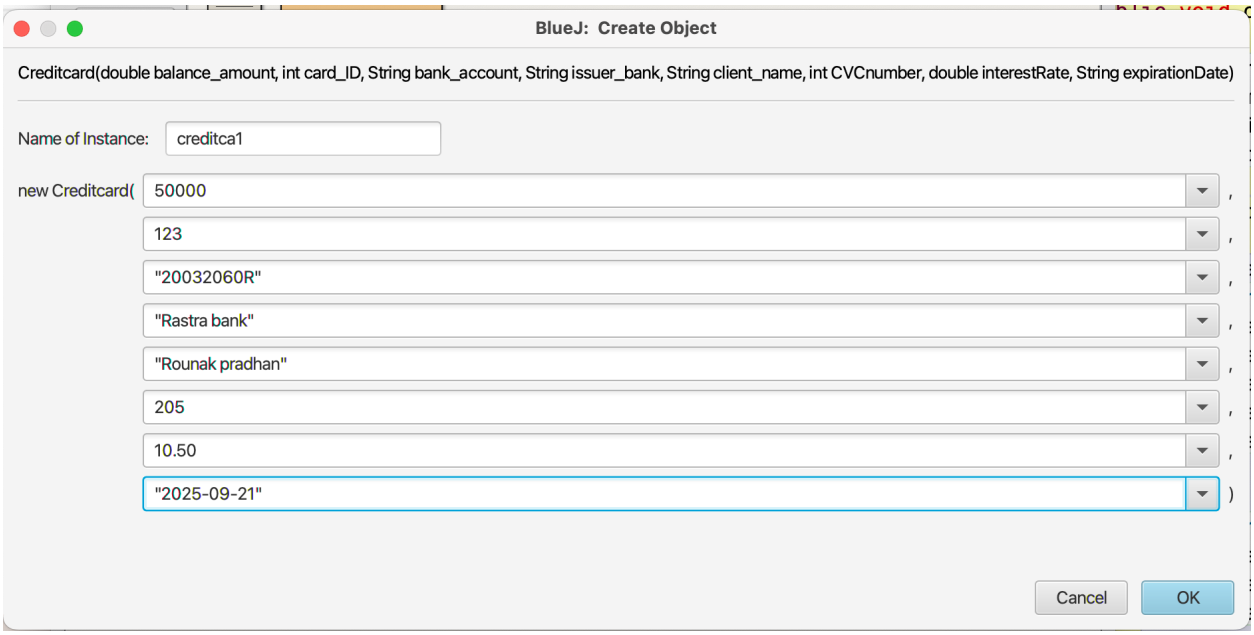

BlueJ: 2066975Rounak Pradhan
New Class...
Compile
Teamwork
Share...
Testing
Run Tests
recording
End
Bankcard
Creditcard
BlueJ: Method Call
void setcreditLimit(double newcreditLimit, int newgracePeriod)
creditca1.setcreditLimit(90000 ,
1)
Cancel
OK
creditca1: Creditcard
creditca1 : Creditcard


creditca1 : Creditcard
private int cvcNumber
205
private double creditLimit
90000.0
private double interestRate
10.5
private String expirationDate
"2025-09-21"
private int gracePeriod
1
private boolean isGranted
true
private int card_ID
123
private String client_name
"Rounak pradhan"
private String issuer_bank
"Rastra bank"
private String bank_account
"20032060R"
private double balance_amount
50000.0
Inspect
Get
Show static fields
Close
Creditcard

5.4. Test 3

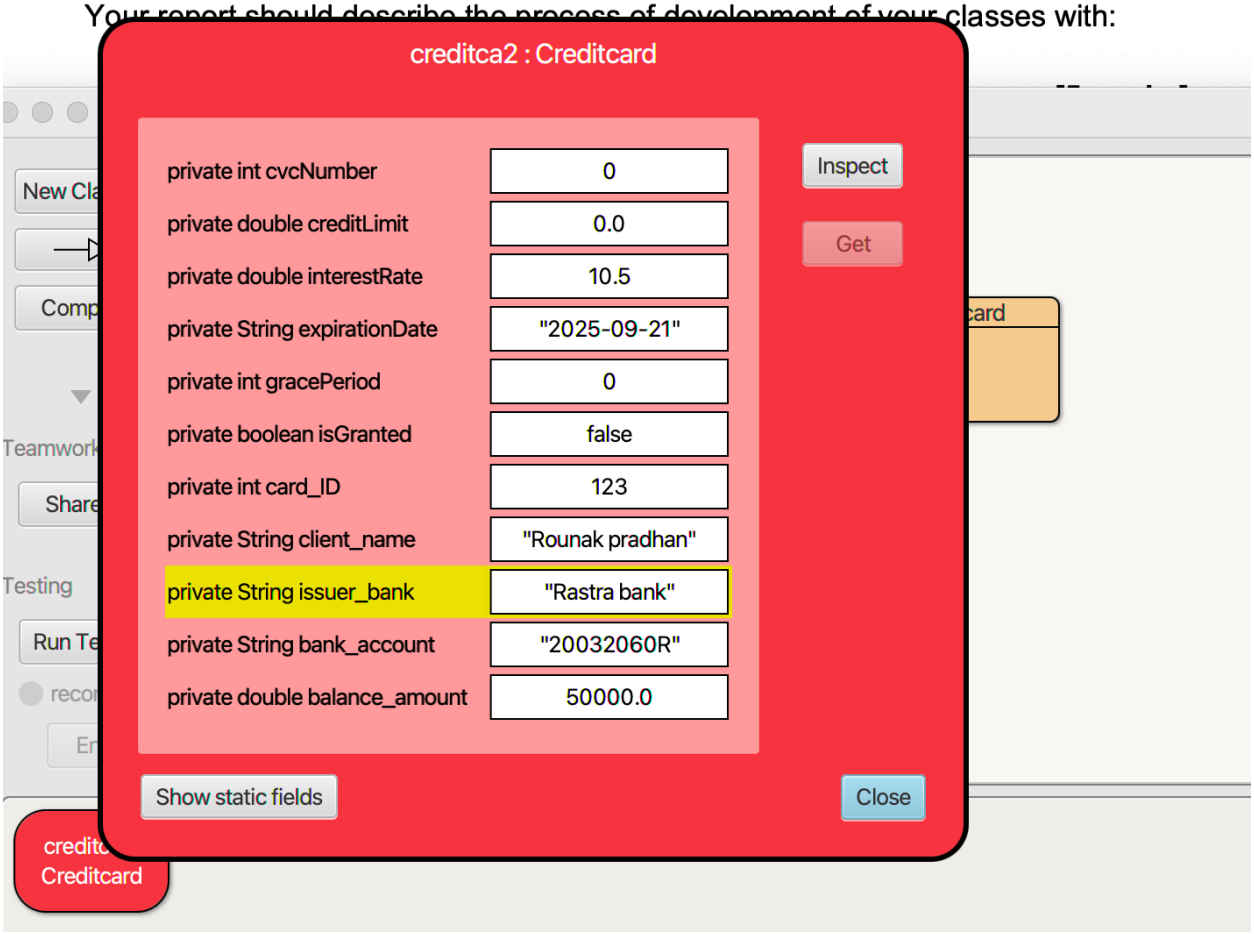
Test number	3
Objective	To inspect the Creditcard class again after cancelling the credit card.
Actions	Creditcard is called with the enlisted arguments: Balance_amount:50000 Card_id:123 Bank_account:”20032060R” Issuer_bank:”Rastra bank” client_name:”Rounak Pradhan” cvcNumber:205 interestRate:10.5 expirationDate:”2025-09-21” Inspection of Creditcard Void cancelcreditcard was called
Expected result	Credit card would be cancelled
Actual result	Credit limit was cancelled
Conclusion	The test was successful.

Table 3 Test3


BlueJ: Create Object
Creditcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int CVCnumber, double interestRate, String expirationDate)
Name of Instance:
creditca1
new Creditcard(
50000
123
"20032060R"
"Rastra bank"
"Rounak pradhan"
205
10.50
"2025-09-21"
Cancel
OK


creditca1 : Creditcard
private int cvcNumber
205
private double creditLimit
90000.0
private double interestRate
10.5
private String expirationDate
"2025-09-21"
private int gracePeriod
1
private boolean isGranted
true
private int card_ID
123
private String client_name
"Rounak pradhan"
private String issuer_bank
"Rastra bank"
private String bank_account
"20032060R"
private double balance_amount
50000.0
Inspect
Get
Show static fields
Close
Creditcard

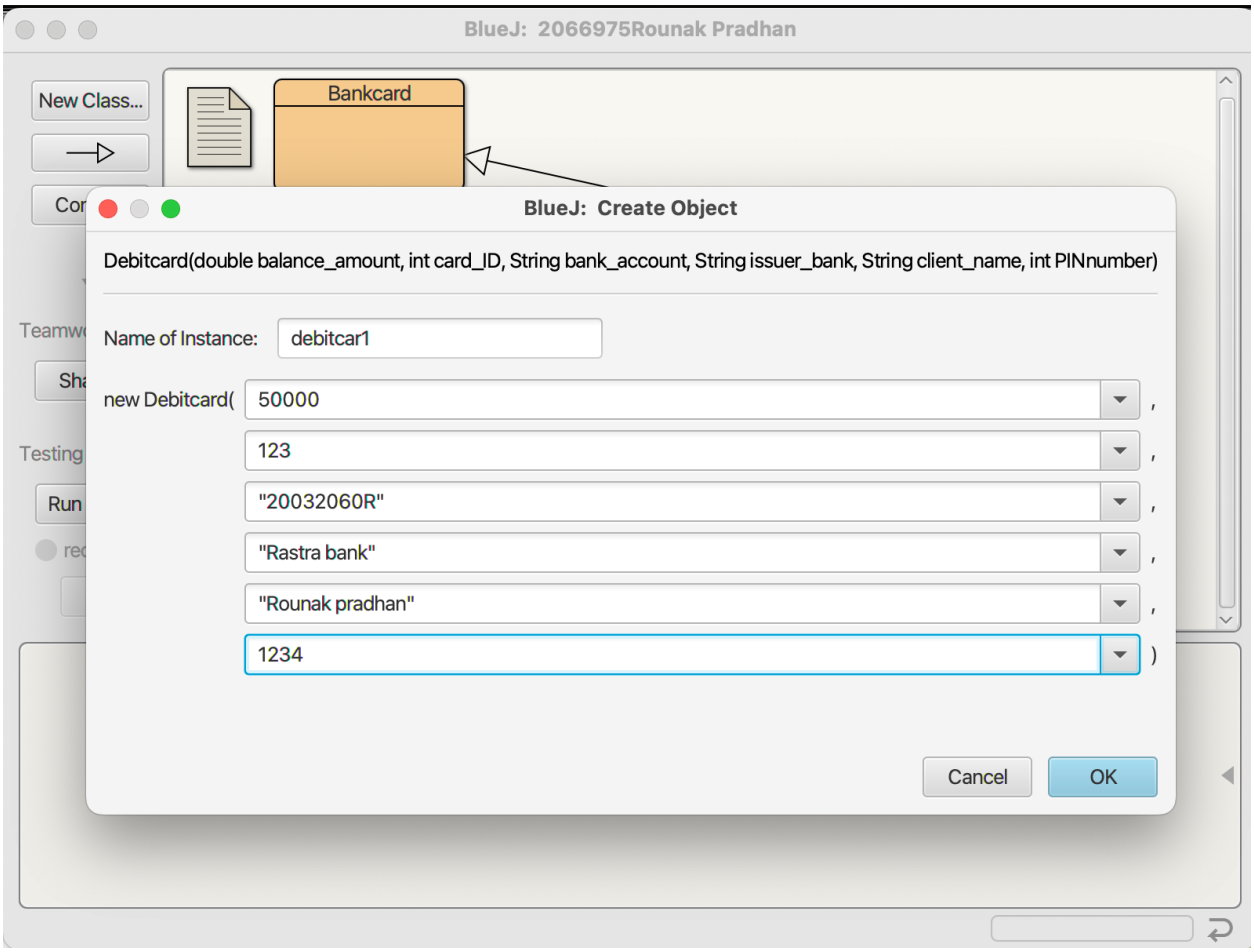
Your report should describe the process of development of your classes with:

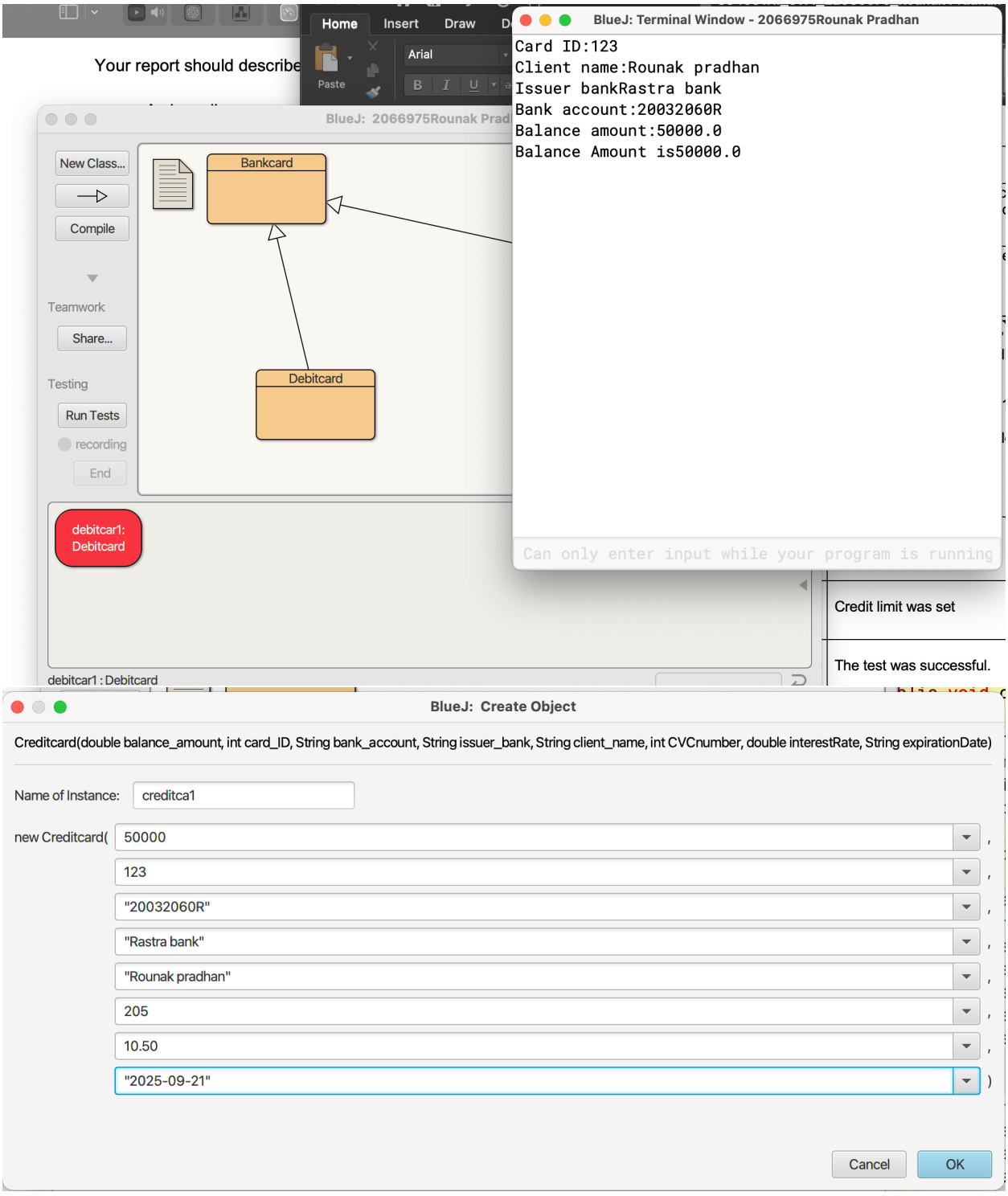


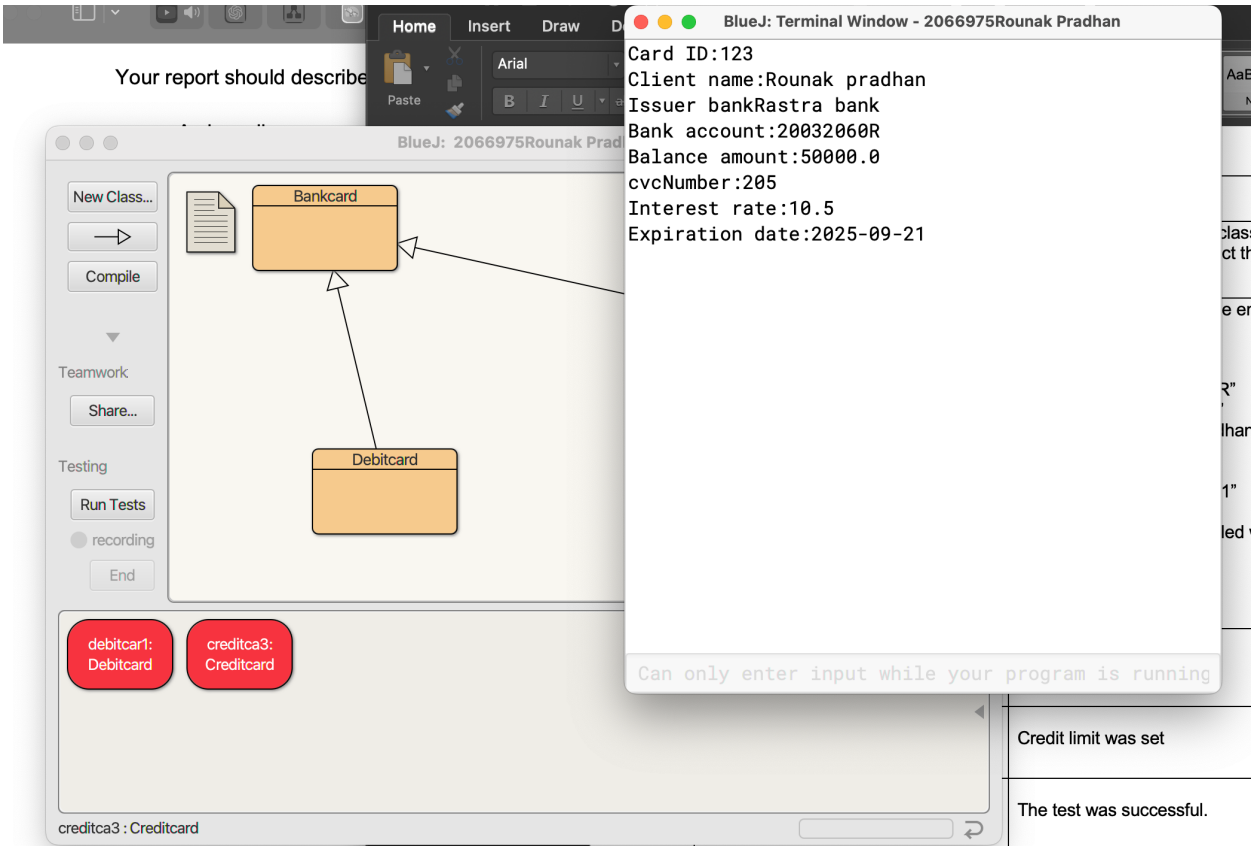
5.4. Test 4

Test number	4
Objective	To display the details of Debit card and Credit card classes.
Actions	void display was called from Debit card void display was called from Credit card
Expected result	Details of Debit card and credit card would be displayed.
Actual result	Details of Credit card and Debit card was displayed.
Conclusion	The test was successful.

Table 4 Test4


BlueJ: 2066975Rounak Pradhan
New Class...
Bankcard
BlueJ: Create Object
Debitcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int PINnumber)
Name of Instance:
debitcar1
new Debitcard(
50000
123
"20032060R"
"Rastra bank"
"Rounak pradhan"
1234
Cancel
OK


Your report should describe
Home
Insert
Draw
Paste
Arial
BlueJ: 2066975Rounak Pradh
New Class...
Compile
Teamwork
Share...
Testing
Run Tests
recording
End
Bankcard
Debitcard
debitcar1: Debitcard
debitcar1 : Debitcard
BlueJ: Terminal Window - 2066975Rounak Pradhan
Card ID:123
Client name:Rounak pradhan
Issuer bankRastra bank
Bank account:20032060R
Balance amount:50000.0
Balance Amount is50000.0
Can only enter input while your program is running
Credit limit was set
The test was successful.
BlueJ: Create Object
Creditcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int CVCnumber, double interestRate, String expirationDate)
Name of Instance:
creditca1
new Creditcard(
50000
123
"20032060R"
"Rastra bank"
"Rounak pradhan"
205
10.50
"2025-09-21"
Cancel
OK


Your report should describe
Home
Insert
Draw
Paste
Arial
BlueJ: Terminal Window - 2066975Rounak Pradhan
Card ID:123
Client name:Rounak pradhan
Issuer bankRastra bank
Bank account:20032060R
Balance amount:50000.0
cvcNumber:205
Interest rate:10.5
Expiration date:2025-09-21
Can only enter input while your program is running
BlueJ: 2066975Rounak Pradh
New Class...
Compile
Teamwork
Share...
Testing
Run Tests
recording
End
Bankcard
Debitcard
debitcar1: Debitcard
creditca3: Creditcard
creditca3 : Creditcard
Credit limit was set
The test was successful.

6.Error detection and correction

Error is user-performed illegal action that cause the application to operate abnormally.

6.1. Syntax Error

In this case semicolon is missing.

```
public String getbank_account()
{
    return bank_account;
}
public double getbalance_amount()
{
    //syntax error
    return balance_amount
}
```

Now Semicolon is added and error is solved.

```
public String getbank_account()
{
    return bank_account;
}
public double getbalance_amount()
{
    //now the syntax error is solved
    return balance_amount;
}
```

6.2. Semantic Error

Semantic error because of wrong data type.

```
public class Bankcard
  {
    private int card_ID;
    private String client_name;
    private int issuer_bank;//Semantic error due wrong data type
    private String bank_account;
    private double balance_amount;
    public Bankcard(double balance_amount,int card_ID,String bank_account,String issuer_bank )
    {
    this.card_ID=card_ID;
    this.client_name="";
    this.issuer_bank=issuer_bank;
```

```
public class Bankcard
  {
    private int card_ID;
    private String client_name;
    private String issuer_bank;//Semantic error solved by changing int to String
    private String bank_account;
    private double balance_amount;
    public Bankcard(double balance_amount,int card_ID,String bank_account,String issuer_bank )
    {
    this.card_ID=card_ID;
    this.client_name="";
    this.issuer_bank=issuer_bank;
    this.bank_account=bank_account;
    this.balance_amount=balance_amount;
```

Semantic solved datatype of issuer_bank is changed from int to String.

6.3.Logical Error

```
public void setcreditLimit(double newcreditLimit, int newgracePeriod)
{
    //logical error
    if (newcreditLimit == 2.5 * getbalance_amount()){
        this.creditLimit= newcreditLimit;
        this.gracePeriod= newgracePeriod;
        this.isGranted= true;
    }
    else{
        System.out.println("Credit cannot be issued");
    }
}
```

```java
public void setcreditLimit(double newcreditLimit, int newgracePeriod)
{
    //logical error is solved
    if (newcreditLimit <= 2.5 * getbalance_amount()){
        this.creditLimit= newcreditLimit;
        this.gracePeriod= newgracePeriod;
        this.isGranted= true;
    }
    else{
        System.out.println("Credit cannot be issued");
    }
}
```

Logical error is solved == to <=

Conclusion

To be honest It takes a lot of effort and commitment to finish this coursework. It is difficult to initially implement scratch of codes in a problem of real world scenario. Starting this activity from scratch makes it seem like there is a ton of material that needs to be mastered while putting in full time effort. I started to feel synced and up to date as I got comfortable with my coding. I came to the conclusion that we can quickly address any significant issues using OOP without experiencing excessive tension. Using OOP is the ideal required method for solving problems in the real world.

After doing this coursework, I feel like I can now play with any OOP concept after completing this program. For the purpose of preventing code retyping, I learned how to use constructors and methods. The problem may be divided into classes and objects quite easily. Last but not least, I value this coursework because it will help me in my job and academic life.

First of all, I struggled with getter and setter methods. Our lecture lecture's frequent assistance helps me keep up with at my own pace. In addition, I utilized a variety of learning tools like ai tools, google and many more. After making so many mistakes, different attempts were attempted to finish the job.

Appendix

Bankcard class code:

```
public class Bankcard
 {
   private int card_ID;
   private String client_name;
   private String issuer_bank;//Semantic error solved by changing int to String
   private String bank_account;
   private double balance_amount;
   public Bankcard(double balance_amount,int card_ID,String bank_account,String
issuer_bank )
   {
   this.card_ID=card_ID;
   this.client_name="";
   this.issuer_bank=issuer_bank;
   this.bank_account=bank_account;
   this.balance_amount=balance_amount;
   }
   public int getcard_Id()
   {
      return card_ID;
   }
   public String getclient_name()
   {
      return client_name;
   }
   public String getissuer_bank()
   {
```

```
        return issuer_bank;
    }
    public String getbank_account()
    {
        return bank_account;
    }
    public double getbalance_amount()
    {
        //now the syntax error is solved
        return balance_amount;
    }
    public void setclient_name(String clientName)
    {
        this.client_name=clientName;
    }
    public void setbalance_amount(double balance_Amount)
    {
        this.balance_amount=balance_Amount;
    }
    public void display()
    {
        System.out.println("Card ID:"+ this.card_ID);
        if (this.client_name. isEmpty()){
            System.out.println("Client name: Not assigned");

        }
        else{
            System.out.println("Client name:"+ client_name);
```

```
        }
        System.out.println("Issuer bank"+ issuer_bank);
        System.out.println("Bank account:"+ bank_account);
        System.out.println("Balance amount:"+ balance_amount);
    }
    }
```

Debitcard class code:

```
public class Debitcard extends Bankcard
{
    private int PINnumber;
    private int WithdrawalAmount;
    private String DateofWithdrawal;
    private boolean hasWithdrawn;


    public Debitcard(double balance_amount, int card_ID, String bank_account, String
issuer_bank, String client_name, int PINnumber)
    {
        super(balance_amount, card_ID, bank_account, issuer_bank);
        super.setclient_name(client_name);
        this.PINnumber= PINnumber;
        this.hasWithdrawn= false;
    }
    public int getPINnumber()
    {
        return this.PINnumber;
    }
    public int getWithdrawalAmount()
    {
        return this.WithdrawalAmount;
    }
    public String getDateofWithdrawal()
    {
        return this.DateofWithdrawal;
    }
    public boolean gethasWithdrawn()
```

```
    {
        return this.hasWithdrawn;
    }
    //mutator method
    public void setWithdrawalAmount(int WithdrawalAmount)
    {
        this.WithdrawalAmount = WithdrawalAmount;
    }
    public void withdraw(int WithdrawalAmount, String dateofWithdrawal, int enteredPin)
    {
        if(enteredPin == PINnumber){
            if(WithdrawalAmount <= getbalance_amount()){
                setbalance_amount(getbalance_amount() - WithdrawalAmount);
                this.DateofWithdrawal= DateofWithdrawal;
                this.WithdrawalAmount=WithdrawalAmount;
                this.hasWithdrawn= true;
                System.out.println("Withdrawal successful");
                System.out.println("New balance amount is"+super.getbalance_amount());

            }
            else {
                System.out.println("Insufficient balance");
                System.out.println("Your Balance Amount is"+super.getbalance_amount());
            }
            }
            else {
                System.out.println("Invalid pin number");
            }
```

```
    }
  public void display()
  {
        super.display();
         if(this.hasWithdrawn == true) {
        System.out.println("the pin number is"+getPINnumber());
        System.out.println("The withdrawal amount is"+getWithdrawalAmount());
        System.out.println("Date of Withdrawal:"+DateofWithdrawal);
        }
         else {
            System.out.println("Balance Amount is"+super.getbalance_amount());
         }
  }
  }
```

Creditcard class code:

```
public class Creditcard extends Bankcard
{
    private int cvcNumber;
    private double creditLimit;
    private double interestRate;
    private String expirationDate;
    private int gracePeriod;
    private boolean isGranted;
    public Creditcard(double balance_amount, int card_ID, String bank_account, String issuer_bank, String client_name, int cvcNumber, double interestRate, String expirationDate)
    {
        super(balance_amount, card_ID, bank_account, issuer_bank);
        super.setclient_name(client_name);
        this.cvcNumber= cvcNumber;
        this.interestRate= interestRate;
        this.expirationDate= expirationDate;
        this.isGranted= false;
    }
    public int getcvcNumber()
    {
        return this.cvcNumber;
    }
    public double getcreditLimit()
    {
        return this.creditLimit;
    }
    public double getinterestRate()
```

```
    {
        return this.interestRate;
    }
    public int getgracePeriod()
    {
        return this.gracePeriod;
    }
    public boolean getisGranted()
    {
        return this.isGranted;
    }
    public void setcreditLimit(double newcreditLimit, int newgracePeriod)
    {
        //logical error is solved
        if (newcreditLimit <= 2.5 * getbalance_amount()){
            this.creditLimit= newcreditLimit;
            this.gracePeriod= newgracePeriod;
            this.isGranted= true;
        }
        else{
            System.out.println("Credit cannot be issued");
        }
    }
        public void cancelCreditcard()
        {
            cvcNumber= 0;
            creditLimit= 0;
            gracePeriod= 0;
```

```
        isGranted= false;
    }
    public void display()
    {
        if (isGranted == true) {
            super.display();
            System.out.println("CVC number:"+this.cvcNumber);
            System.out.println("Interest rate:"+this.interestRate);
            System.out.println("Expiration date:"+this.expirationDate);
            System.out.println("Credit limit:"+this.creditLimit);
            System.out.println("Grace period:"+this.gracePeriod);
        }
        else {
            super.display();
            System.out.println("cvcNumber:"+this.cvcNumber);
            System.out.println("Interest rate:"+this.interestRate);
            System.out.println("Expiration date:"+this.expirationDate);
        }
    }
}
```