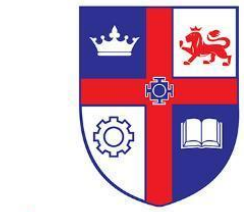
LONDON METROPOLITAN UNIVERSITY



Islington college
(इस्लिङ्टन कलेज)

# CS4001NI Programming

## 30% Individual Coursework

## 2022-23 Autumn


**Student Name: Aadesh Shrestha**

**London Met ID: 22067566**

**College ID:** NP01CP4A220063

**Group: C3**


**Assignment Due Date: Friday, January 27, 2023**

**Assignment Submission Date: Friday, January 27, 2023**

*I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.*

## Table of Contents

1. Introduction ..... 1

1.1 About Project ..... 1

1.2 Tools Used ..... 1

2. Class Diagram ..... 2

2.1 BankCard ..... 2

2.2 DebitCard ..... 3

2.3 CreditCard ..... 4

2.4 Inheritance ..... 5

3. Pseudocode ..... 6

3.1 Pseudocode of BankCard ..... 6

3.2 Pseudocode of DebitCard ..... 9

3.3 Pseudocode of CreditCard ..... 13

4. Description of Methods ..... 17

4.1 For BankCard class ..... 17

4.2 For DebitCard class ..... 19

4.3 For CreditCard class ..... 21

5. Testing ..... 23

5.1 Test 1: Inspect the Debit Card class, withdraw the amount, and re-inspect the Debit Card Class. ..... 23

Output Result: ..... 24

5.2 Test 2: Inspect Credit Card class, set the credit limit and reinspect the Credit Card class. ..... 28

Output Result: ..... 29

5.3 Test 3: Inspect Credit Card class again after cancelling the credit card. ..... 33

Output Result: ..... 34

5.4 Test 4: Display the details of Debit Card and Credit Card classes. 36
Output Result: 36
6. Error Detection and Correction 38
6.1 Error 1 38
6.2 Error 2 38
6.3 Error 3 39
6.4 Types of Errors 40
7. Conclusion 41
8. References 42
9. Appendix 43
9.1 Appendix for BankCard class 43
9.2 Appendix for DebitCard class 47
9.2 Appendix for CreditCard class 52

## List of Figure

Figure 1: Class diagram of BankCard ..... 2
Figure 2: Class diagram of DebitCard ..... 3
Figure 3: Class diagram of CreditCard ..... 4
Figure 4: Class diagram of Inheritance ..... 5
Figure 5: Screenshot of assigning the data in the Debit Card class ..... 24
Figure 6: Screenshot of the inspection of the Debit Card class ..... 25
Figure 7: Screenshot of assigning the data for void withdraw ..... 26
Figure 8: Screenshot of re-inspection of the Debit Card class ..... 27
Figure 9: Screenshot of assigning the data in the Credit Card class ..... 29
Figure 10: Screenshot of the inspection of the Credit Card class ..... 30
Figure 11: Screenshot of assigning the data for void setCreditLimit ..... 31
Figure 12: Screenshot of re-inspection of the Credit Card class ..... 32
Figure 13: Screenshot of inspection of the Credit Card class ..... 34
Figure 14: Screenshot of re-inspection of the Credit Card class after calling cancelCreditCard method ..... 35
Figure 15: Screenshot for displaying details of Debit Card class ..... 36
Figure 16: Screenshot for displaying details of Credit Card class ..... 37
Figure 17: Error 1 ..... 38
Figure 18: Correction of error 1 ..... 38
Figure 19: Error 2 ..... 38
Figure 20: Error 2 ..... 38
Figure 21: Correction of error 2 ..... 39
Figure 22: Error 3 ..... 39
Figure 23: Correction of error 3 ..... 39

## List of Table

Table 1:To inspect the Debit Card class, withdraw the amount, and re-inspect the Debit Card Class ..... 23
Table 2: To inspect Credit Card class, set the credit limit and reinspect the Credit Card class ..... 28
Table 3: To inspect Credit Card class again after cancelling the credit card ..... 33
Table 4: To display the details of Debit Card and Credit Card classes ..... 36

# 1. Introduction

## 1.1 About Project

The objective of this Java project is to implement a real-world scenario with the application of Object-oriented programming in Java. In this project, the 3 created classes are: BankCard class, DebitCard class and CreditCard class in which BankCard is the parent class whereas DebitCard and CreditCard are the child classes, forming a relationship called hierarchical relationship. Some features of Java utilized in this project are constructor, getter, setter, logical expression and operators etc.

BankCard class is class which consists of various bank details as inputs (i.e cardID, issuerBank, clientName, bankAccount and balanceAmount) and each attribute has their own accessor methods. It sets the value for client name and balance amount of a user. It also has a method to display all the bank details.

DebitCard is a sub class which takes same bank details of BankCard class as inputs with the addition of other attributes (i.e pinNumber, withdrawalAmount, dateOfWithdraw and hasWithdrawn) and each attribute has their own accessor methods. It sets the value for withdrawalAmount. It consists of two methods one to withdraw cash from the card if certain condtions are fulfilled and the other to display the content of the card.

CreditCard is a sub class which takes same bank details of BankCard class as inputs with the addition of other attributes (i.e are cvcNumber, creditLimit, interestRate, expirationDate, gracePeriod and isGranted) and each attribute has their own accessor methods. It also sets the value of creditLimit and gracePeriod if certain condition is satisfied. It has two methods one to cancel the credit card and the other to display the content of the card.

## 1.2 Tools Used

BlueJ was used to write the codes for this project. BlueJ was preferred to other IDEs because it is a development environment which is interactive, portable, innovative and simple to use. (BlueJ org, n.d.) Draw.io was used to create the class diagrams. Finally, Microsoft word was used for the documentation of this project.

## 2. Class Diagram

### 2.1 BankCard

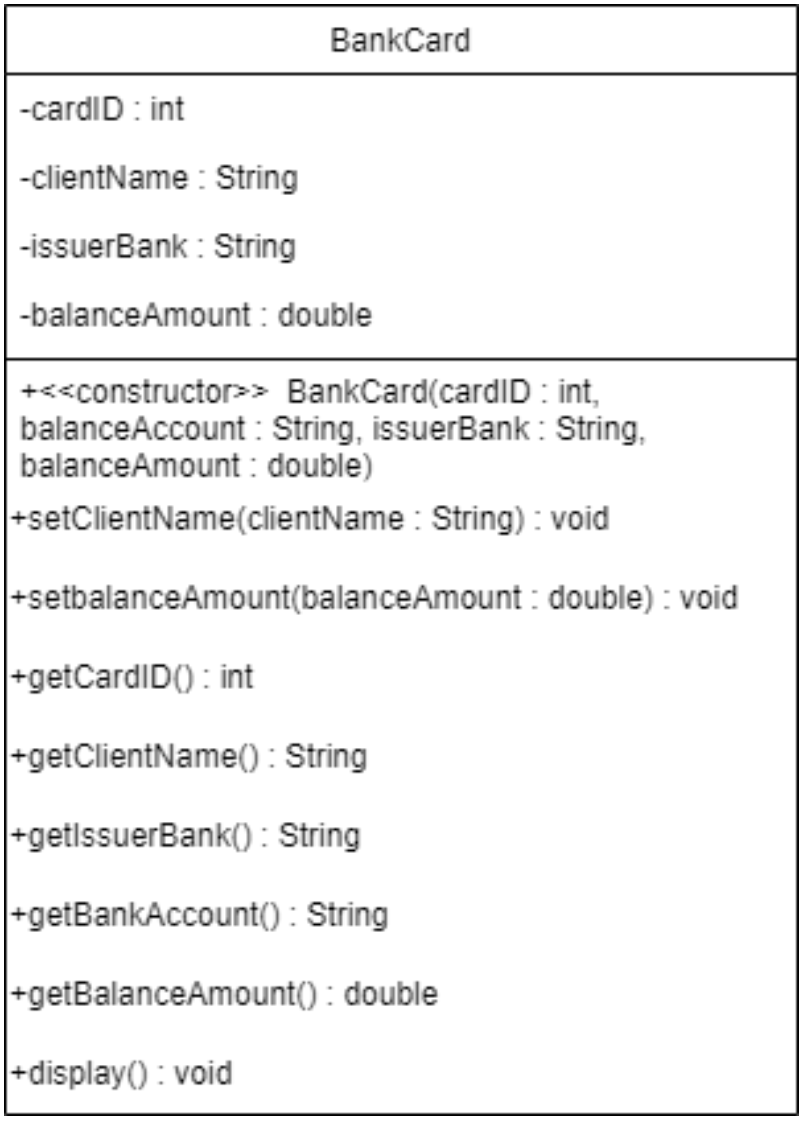



*Figure 1: Class diagram of BankCard*

### 2.2 DebitCard


*Figure 2: Class diagram of DebitCard*

### 2.3 CreditCard

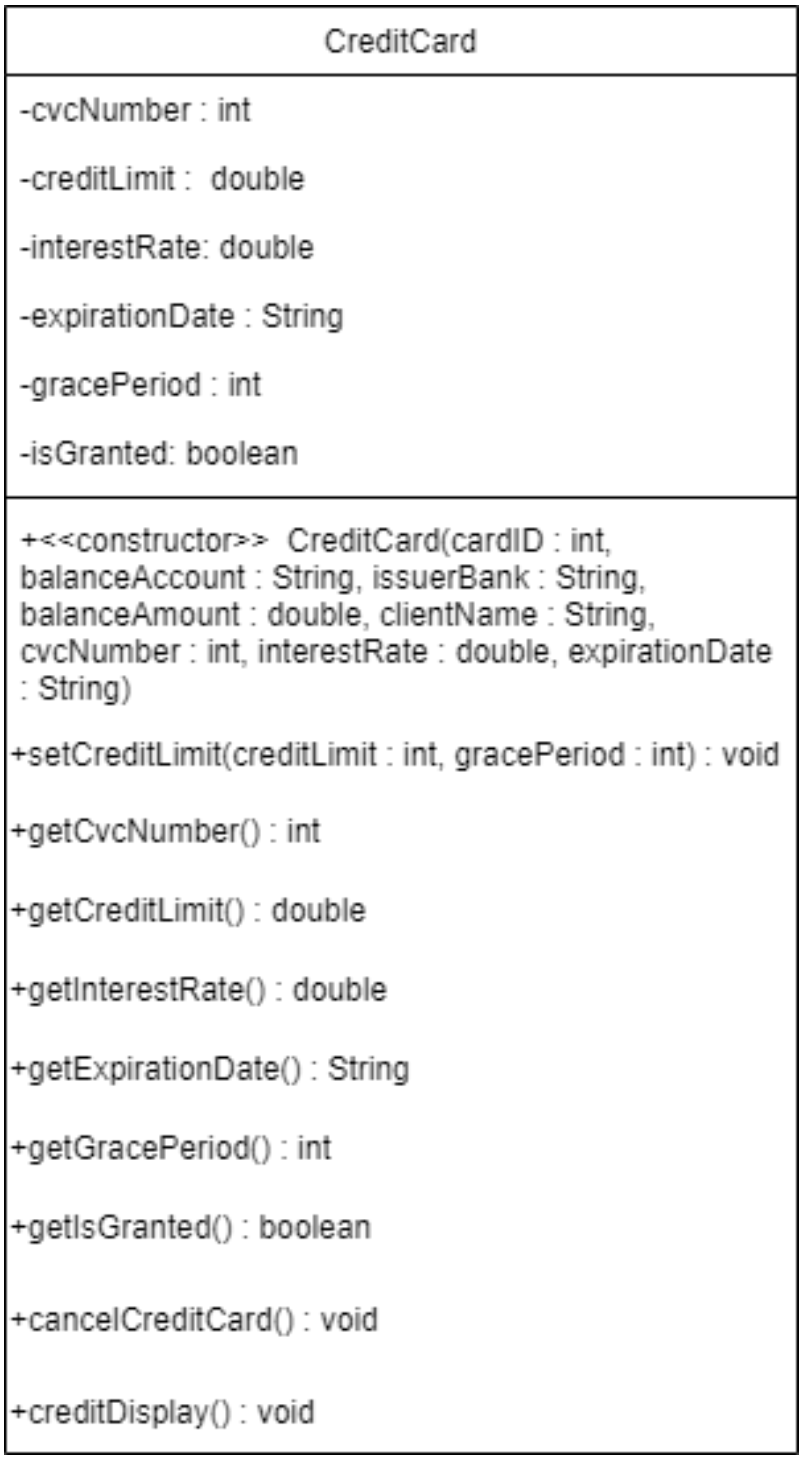



*Figure 3: Class diagram of CreditCard*

## 2.4 Inheritance

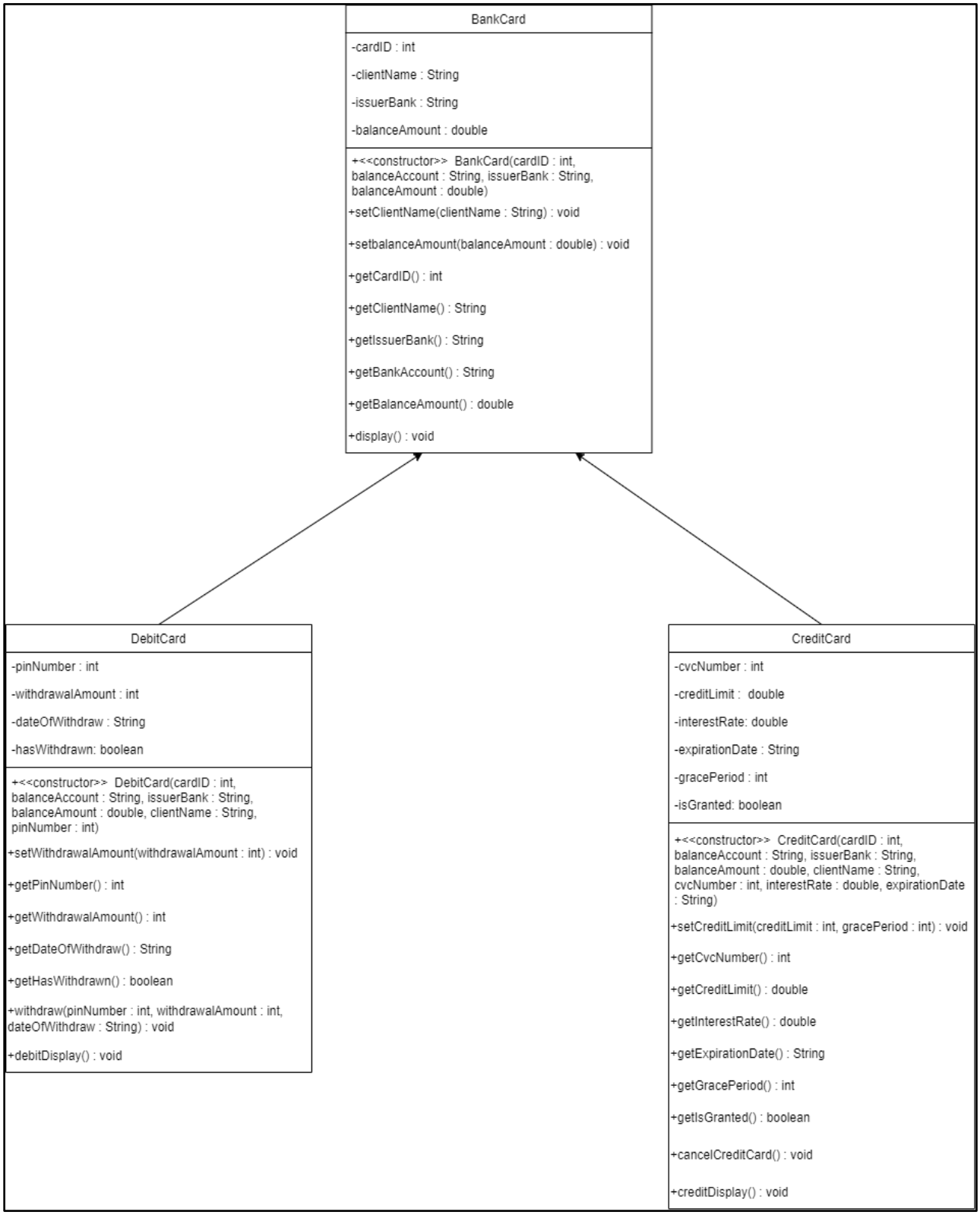



*Figure 4: Class diagram of Inheritance*

# 3. Pseudocode

## 3.1 Pseudocode of BankCard

**CREATE** class BankCard

**DECLARE** attribute cardId as integer

**DECLARE** attribute clientName as String

**DECLARE** attribute issuerBank as String

**DECLARE** attribute bankAccount as String

**DECLARE** attribute balanceAmount as number

**CREATE** constructor BankCard (PASS parameters cardID as integer, issuerBank as String, bankAccount as String, balancaeAmount as number)

**DO**

**ASSIGN** value of parameter cardID to attr. cardID

**ASSIGN** value of parameter issuerBank to attr. issuerBank

**ASSIGN** value of parameter bankAccount to attr. bankAccount

**ASSIGN** value of parameter balanceAmount to attr. balanceAmount

**ASSIGN** value of attr. clientName to empty.

**END DO**

**CREATE** a mutator method setClientName **(PASS** parameter clientName as String)

**DO**

**ASSIGN** value of parameter clientName to attr. of clientName

**END DO**

**CREATE** a mutator method setBalanceAmount (**PASS** parameter balanceAmount as number)

**DO**

**ASSIGN** value of parameter balanceAmount to attr. of balanceAmount

**END DO**

**CREATE** an accessor method getCardID() with returntype integer

**DO**

**RETURN** cardID

**END DO**

**CREATE** an accessor method getClientName() with returntype String

**DO**

**RETURN** clientName

**END DO**

**CREATE** an accessor method getIssuerBank() with returntype String

**DO**

**RETURN** issuerBank

**END DO**

**CREATE** an accessor method getBankAccount() with returntype String

**DO**

    **RETURN** bankAccount

**END DO**

**CREATE** an accessor method getBalanceAmount() with returntype number

**DO**

    **RETURN** balanceAmount

**END DO**

**CREATE** method display ()

**DO**

    **PRINT** getCardID ()

    **PRINT** getIssuerBank ()

    **PRINT** getBankAccount ()

    **PRINT** getBalanceAmount ()

    **IF** getClientName () != “ ”

        **PRINT** getClientName ()

    **ELSE**

        **PRINT** "Client Name has not been registered."

    **END IF**

**END DO**

### 3.2 Pseudocode of DebitCard

**CREATE** class DebitCard which extends the class BankCard

**DECLARE** attribute pinNumber as integer

**DECLARE** attribute withdrawalAmount as integer

**DECLARE** attribute dateOfWithdraw as String

**DECLARE** attribute hasWithdrawn as Boolean

**CREATE** constructor DebitCard(**PASS** parameters cardID, pinNumber as integer, issuerBank, bankAccount, clientName as String, balanceAmount as number)

**DO**

**CALL** super from parent class BankCard with attributes cardID, issuerBank, bankAccount, balanceAmount

**CALL** method setClientName (clientName)

**ASSIGN** value of parameter pinNumber to attr. pinNumber

**ASSIGN** value of attr. hasWithdrawn to false

**END DO**

**CREATE** a mutator method setWithdrawalAmount (**PASS** parameter withdrwalAmount as integer)

**DO**

**ASSIGN** value of parameter withdrwalAmount to attr. of withdrwalAmount

**END DO**

**CREATE** an accessor method getPinNumber () with returntype integer

**DO**

**RETURN** pinNumber

**END DO**

**CREATE** an accessor method getWithdrawalAmount () with returntype String

**DO**

**RETURN** withdrawalAmount

**END DO**

**CREATE** an accessor method getDateOfWithdraw () with returntype integer

**DO**

**RETURN** dateOfWithdraw

**END DO**

**CREATE** an accessor method getHasWithdraw () with returntype integer

**DO**

**RETURN** hasWithdraw

**END DO**

**CREATE** method withdraw (PASS parameters pinNumber, withdrawalAmount as integer, dateOfWithdraw as String)

**DO**

**IF** value of parameter pinNumber = value of attr. pinNumber

**IF** value of parameter withdrawalAmount < value of getBalacneAmount () and value of parameter withdrawalAmount > 0

**ASSIGN** value of parameter pinNumber to attr. of pinNumber

**ASSIGN** value of parameter withdrwalAmount to attr. of withdrawalAmount

**ASSIGN** value of parameter dateOfWithdraw to attr. of dateOfWithdraw

**ASSIGN** value of hasWithdraw to true

**CALL** method setBalanceAmount from BankCard class (PASS method getBalanceAmount() from BankCard class - withdrawalAmount)

**PRINT** "Your Transaction was Successful."

**ELSE**

**PRINT** “You have Insufficent Balance.”

**END IF**

**ELSE**

**PRINT** "The entered PIN number is incorrect."

**END IF**

**END DO**

**CREATE** method debitDisplay ()

**DO**

**IF** hasWithdrawn = true

**CALL** method display () from BankCard class

**PRINT** getPinNumber ()

**PRINT** getWithdrawalAmount ()

**PRINT** getDateOfWithdraw ()

**ELSE**

**PRINT** getBalanceAmount () from BankCard class

**END IF**

**END DO**

### 3.3 Pseudocode of CreditCard

**CREATE** class CreditCard which extends the class BankCard

**DECLARE** attribute cvcNumber as integer

**DECLARE** attribute creditLimit as number

**DECLARE** attribute interestRate as number

**DECLARE** attribute expirationDate as String

**DECLARE** attribute gracePeriod as integer

**DECLARE** attribute isGranted as Boolean

**CREATE** constructor CreditCard(**PASS** parameters cardID, cvcNumber as integer, issuerBank, bankAccount, clientName, expirationDate as String, balanceAmount, interestRate as number)

**DO**

**CALL** super from parent class BankCard with attributes cardID, issuerBank, bankAccount, balanceAmount

**CALL** method setClientName (clientName)

**ASSIGN** value of parameter cvcNumber to attr. cvcNumber

**ASSIGN** value of parameter interestRate to attr. interestRate

**ASSIGN** value of parameter expirationDate to attr. expirationDate

**ASSIGN** value of attr. isGranted to false

**END DO**

**CREATE** a mutator method setCreditLimit (**PASS** parameter creditLimit, gracePeriod as integer)

**DO**

**IF** value of parameter creditLimit <= getBalanceAmount () multiplied by 2.5

**ASSIGN** value of parameter creditLimit to attr. creditLimit

**ASSIGN** value of parameter gracePeriod to attr. gracePeriod

**ASSIGN** value of attr. isGranted to true

**ELSE**

**PRINT** "Your requested Amount exceeds the Credit Limit."

**END IF**

**END DO**

**CREATE** an accessor method getCvcNumber () with returntype integer

**DO**

**RETURN** cvcNumber

**END DO**

**CREATE** an accessor method getCreditLimit () with returntype number

**DO**

**RETURN** creditLimit

**END DO**

**CREATE** an accessor method getInterestRate () with returntype number

**DO**

**RETURN** interestRate

**END DO**

**CREATE** an accessor method getExpirationDate () with returntype String

**DO**

**RETURN** expiraitonDate

**END DO**

**CREATE** an accessor method getGracePeriod () with returntype integer

**DO**

**RETURN** gracePeriod

**END DO**

**CREATE** an accessor method getIsGranted () with returntype Boolean

**DO**

**RETURN** isGranted

**END DO**

**CREATE** method cancelCreditCancel ()

**DO**

**ASSIGN** value of attr. cvcNumber to 0

**ASSIGN** value of attr. creditLimit to 0

**ASSIGN** value of attr. gracePeriod to 0

**ASSIGN** value of attr. isGranted to false

**END DO**

**CREATE** method creditDisplay ()

**DO**

**IF** isGranted = true

**CALL** method display () from BankCard class

**PRINT** cvcNumber

**PRINT** creditLimit

**PRINT** interestRate

**PRINT** expirationDate

**PRINT** gracePeriod

**ELSE**

**CALL** method display () from BankCard class

**PRINT** cvcNumber

**PRINT** interestRate

**PRINT** expirationDate

**END IF**

**END DO**

# 4. Description of Methods

## 4.1 For BankCard class

The BankCard class contains 5 attributes which are cardID, clientName, issuerBank, bankAccount and balanceAmount along with 9 methods. The methods are as follows:

**Method 1:** BankCard ()

This is a constructor which accepts 4 parameters (cardID, issuerBank, bankAccount, balanceAmount). It is used to initialize the instance variables passed in the parameters. Initially, the value of clientName is set to “ ”.

**Method 2:** setClientName ()

This method is a mutator method whose only purpose is to accept a parameter (clientName) and assigns its value to instance variable clientName.

**Method 3:** setBalanceAmount ()

This method is a mutator method whose only purpose is to accept a parameter (balanceAmount) and assigns its value to instance variable balanceAmount. This method is called several times in sub classes to change its value according to the requirement of the condition.

**Method 4:** getCardID ()

This method is an accessor method whose only purpose is to return the value of cardID and its return type is int.

**Method 5:** getClientName ()

This method is an accessor method whose only purpose is to return the value of clientName and its return type is String.

**Method 6:** getIssuerBank ()

This method is an accessor method whose only purpose is to return the value of issuerBank and its return type is String.

**Method 7:** getBankAccount ()

This method is an accessor method whose only purpose is to return the value of bankAccount and its return type is String.

**Method 8:** getBalanceAmount ()

This method is an accessor method whose only purpose is to return the value of balanceAmount and its return type is double.

**Method 9:** display ()

The function of this method is to displays the values of cardID, issuerBank, bankAccount and balanceAmount. The clientName is also displayed if the condtion (clientName !="") is satisfied. Proper description of the printed values is provided as well.

### 4.2 For DebitCard class

The DebitCard class contains 4 attributes which are pinNumber, withdrawalAmount, dateOfWithdraw and hasWithdrawn along with 8 methods. The methods are as follows:

**Method 1:** DebitCard ()

This is a constructor which accepts 6 parameters (cardID, issuerBank, bankAccount, balanceAmount, clientName, pinNumber). This method calls the attributes (cardID, issuerBank, bankAccount, balanceAmount) from BankCard class and calls the setClientName(clientName) to set the value of attribute clientName. It is used to set values of the instance variables passed in the parameters. The value of hasWithdraw is set to false.

**Method 2:** setWithdrawalAmount ()

This method is a mutator method whose only purpose is to accept a parameter (withdrawalAmount) and assigns its value to instance variable withdrawalAmount.

**Method 3:** getPinNumber ()

This method is an accessor method whose only purpose is to return the value of pinNumber and its return type is int.

**Method 4:** getWithdrawalAmount ()

This method is an accessor method whose only purpose is to return the value of withdrawalAmount and its return type is int.

**Method 5:** getDateOfWithdraw ()

This method is an accessor method whose only purpose is to return the value of dateOfWithdraw and its return type is String.

**Method 6:** getHasWithdrawn ()

This method is an accessor method whose only purpose is to return the value of hasWithdrawn and its return type is boolean.

**Method 7:** withdraw ()

The function of this method is to withdraw certain about of balance from balanceAmount. This method consists of 3 parameters (pinNumber, withdrawalAmount, dateOfWithdraw). Once this method is called, the parameter pinNumber should be equal to the attribute pinNumber and the parameter withdrawalAmount should be less than method getBalanceAmount(). If the condition is fulfilled, then its assigns the value of parameter withdrawalAmount and dateOfWithdraw to attribute withdrawalAmount and dateOfWithdraw respectively. The value of attribute hasWithdrawn is set to true. Also, the new value of balanceAmount is assigned according to the deduction with withdrawalAmount which is accomplished by setBalanceAmount() . But if any one of the conditions is not satisfied then a suitable message will be displayed.

**Method 8:** debitDisplay ()

This method is used to display all the content of DebitCard class in detail. Firstly, it checks the value of hasWithdrawn is equal to true or not. If the condition is satisfied, it calls method display () from BankCard class and prints the value of attributes pinNumber, withdrawalAmount and dateOfWithdraw of DebitCard class with proper description for the viewer to understand it. If the condition is not satisfied, then only the balanceAmount from BankCard class is displayed.

### 4.3 For CreditCard class

The DebitCard class contains 6 attributes which are cvcNumber, creditLimit, interestRate, expirationDate, gracePeriod and isGranted along with 10 methods. The methods are as follows:

**Method 1:** CreditCard ()

This is a constructor which accepts 8 parameters (cardID, issuerBank, bankAccount, balanceAmount, clientName, cvcNumber, interestRate, expiratioDate). This method calls the attributes (cardID, issuerBank, bankAccount, balanceAmount) from BankCard class and calls the setClientName(clientName) to set the value of attribute clientName. It assigns the value of parameter to their respective instance variable. The value of isGranted is set to false.

**Method 2:** setCreditLimit ()

This mutator method accepts creditLimit and gracePeriod as parameters. If the value of parameter creditLimit is lesser or equal to balanceAmount multiplied by 2.5 then the method assigns the value of parameter creditLimit and garcePeriod to instance variable of creditLimit and gracePeriod respectively. If the condition is not fulfilled, a suitable message is displayed.

**Method 3:** getCvcNumber ()

This method is an accessor method whose only purpose is to return the value of cvcNumber and its return type is int.

**Method 4:** getCreditLimit()

This method is an accessor method whose only purpose is to return the value of creditLimit and its return type is double.

**Method 5:** getInterestRate ()

This method is an accessor method whose only purpose is to return the value of interestRate and its return type is double.

**Method 6:** getExpirationDate ()

This method is an accessor method whose only purpose is to return the value of expirationDate and its return type is double.

**Method 7:** getGracePeriod ()

This method is an accessor method whose only purpose is to return the value of gracePeriod and its return type is int.

**Method 8:** getIsGranted ()

This method is an accessor method whose only purpose is to return the value of isGranted and its return type is boolean.

**Method 9:** cancelCreditCard ()

This method is used to remove the credit card feature. When this method is called, it sets the value of attributes cvcNumber, creditLimit and gracePeriod to 0 and the value of attribute isGranted to false.

**Method 10:** creditDisplay ()

This method is used to display all the content of CreditCard class in detail. Firstly, it checks the value of isGranted is equal to true or not. If the condition is satisfied, it calls method display () from BankCard class and prints the value of attributes cvcNumber, creditLimit, interestRate, expirationDate and gracePeriod of CreditCard class with proper description for the viewer to understand it. If the condition is not satisfied, it displays same as when the condition is satisfied but does not print value of creditLimit and gracePeriod.

## 5. Testing

**5.1 Test 1:** Inspect the Debit Card class, withdraw the amount, and re-inspect the Debit Card Class.

| | |
|---|---|
| Test No. | 1 |
| Objective | To inspect the Debit Card class, withdraw the amount, and re-inspect the Debit Card Class. |
| Action | - The Debit Card is called with the following arguments:<br>cardID=1<br>issuerBank=”Everest Bank”<br>bankAccount= "321477556"<br>balanceAmount=10000<br>clientName=”Aadesh Shrestha”<br>pinNumber=1234<br>- Inspection of the Debit Card class<br>- void withdraw() is called with the following arguments:<br>pinNumber=1234<br>withdrawalAmount=5000<br>dateOfWithdraw=”2023-01-03”<br>-Re-inspection of the Debit Card |
| Expected Result | Certain amount of balance would be withdrawn from a debit card. |
| Actual Result | Certain amount of balance was withdrawn. |
| Conclusion | The test is successful. |

*Table 1:To inspect the Debit Card class, withdraw the amount, and re-inspect the Debit Card Class*

## Output Result:

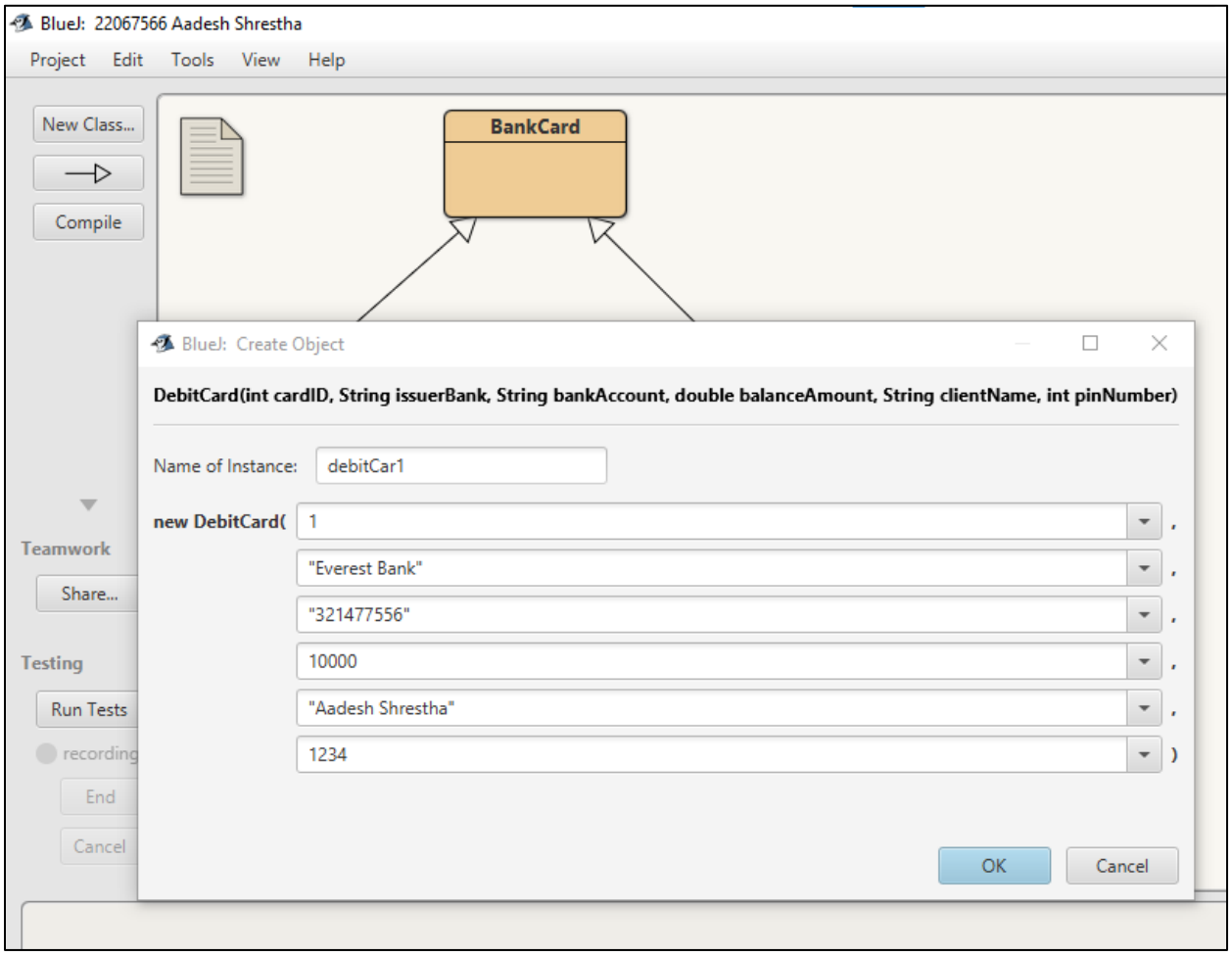



*Figure 5: Screenshot of assigning the data in the Debit Card class*

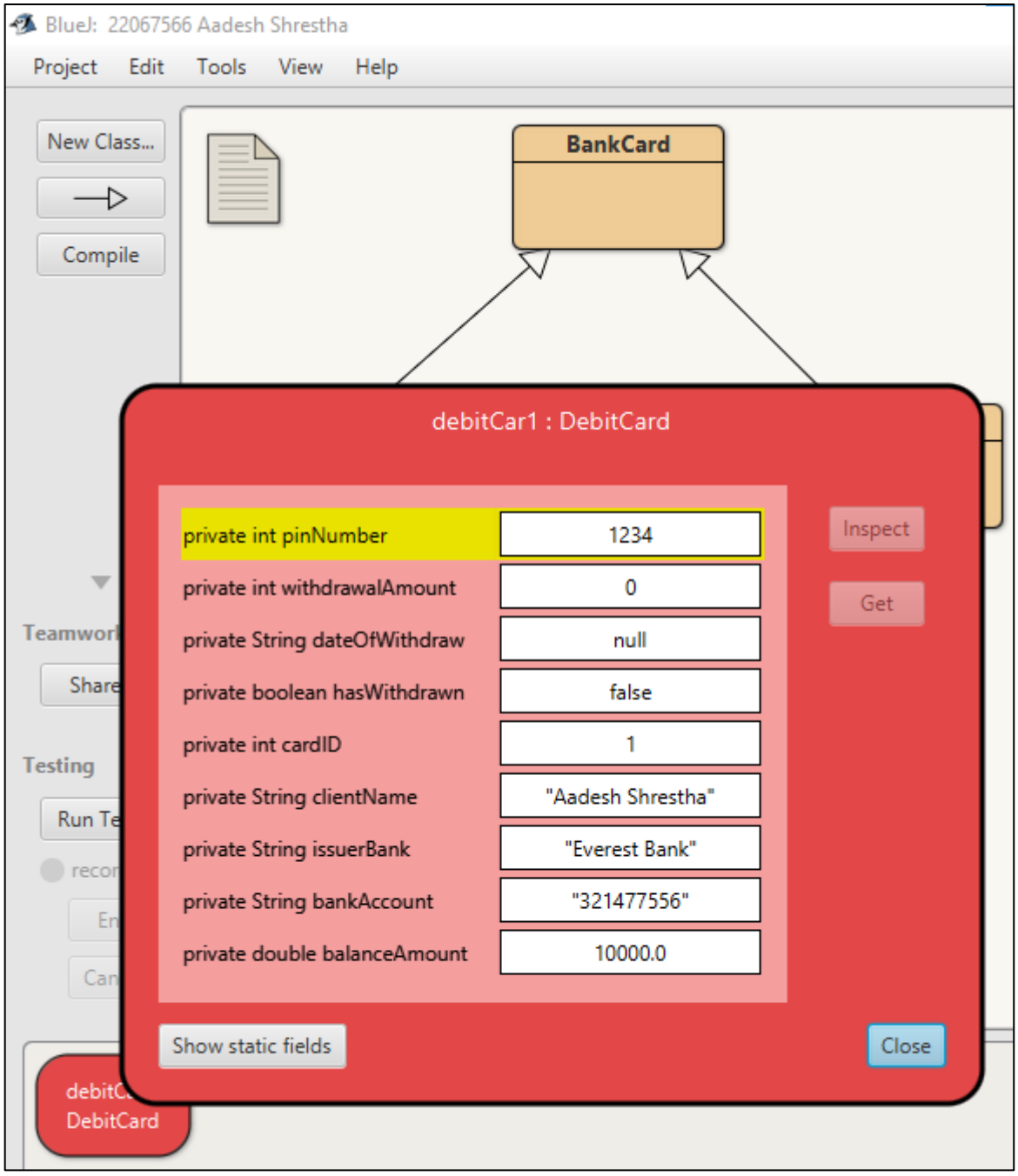



*Figure 6: Screenshot of the inspection of the Debit Card class*

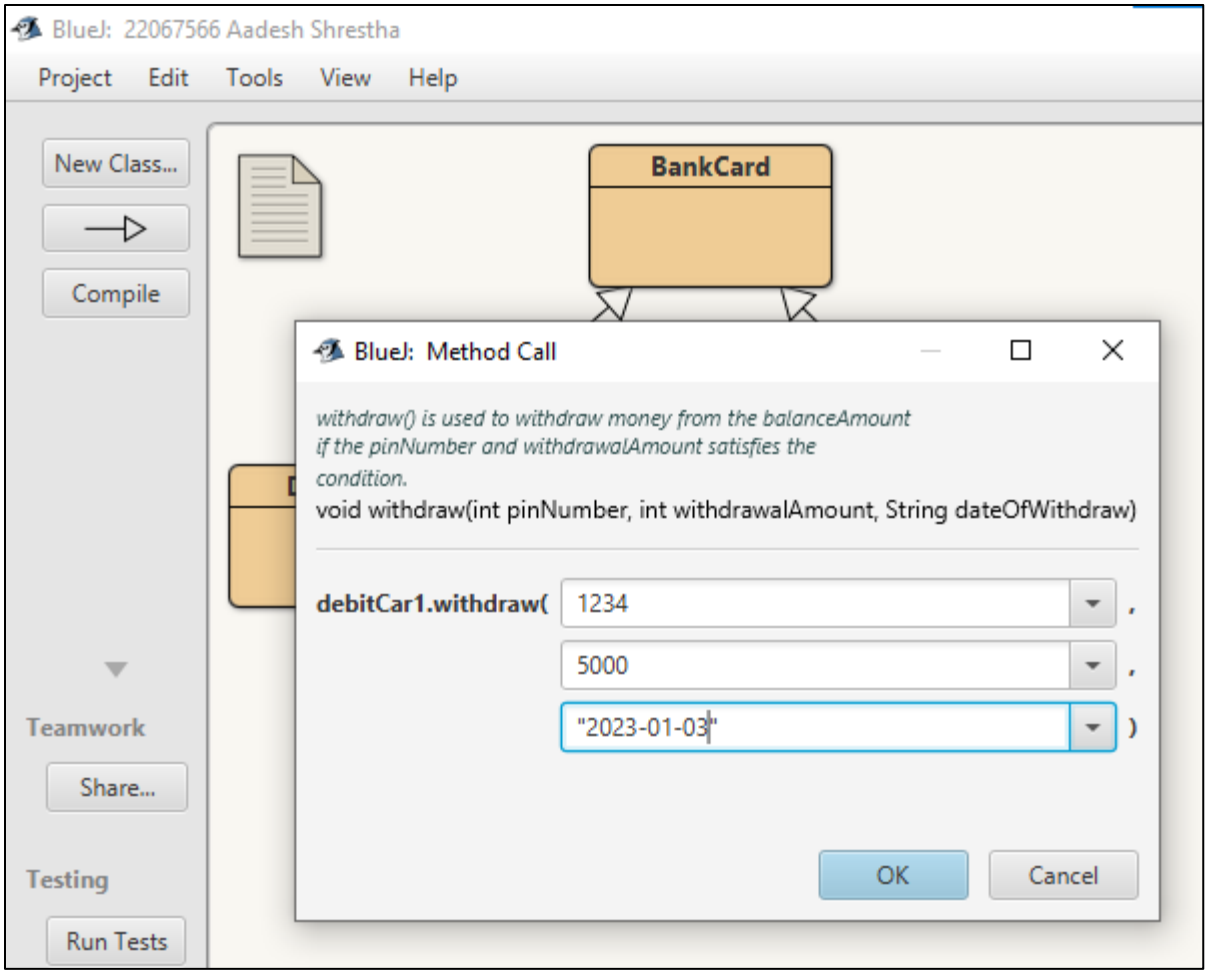



*Figure 7: Screenshot of assigning the data for void withdraw*

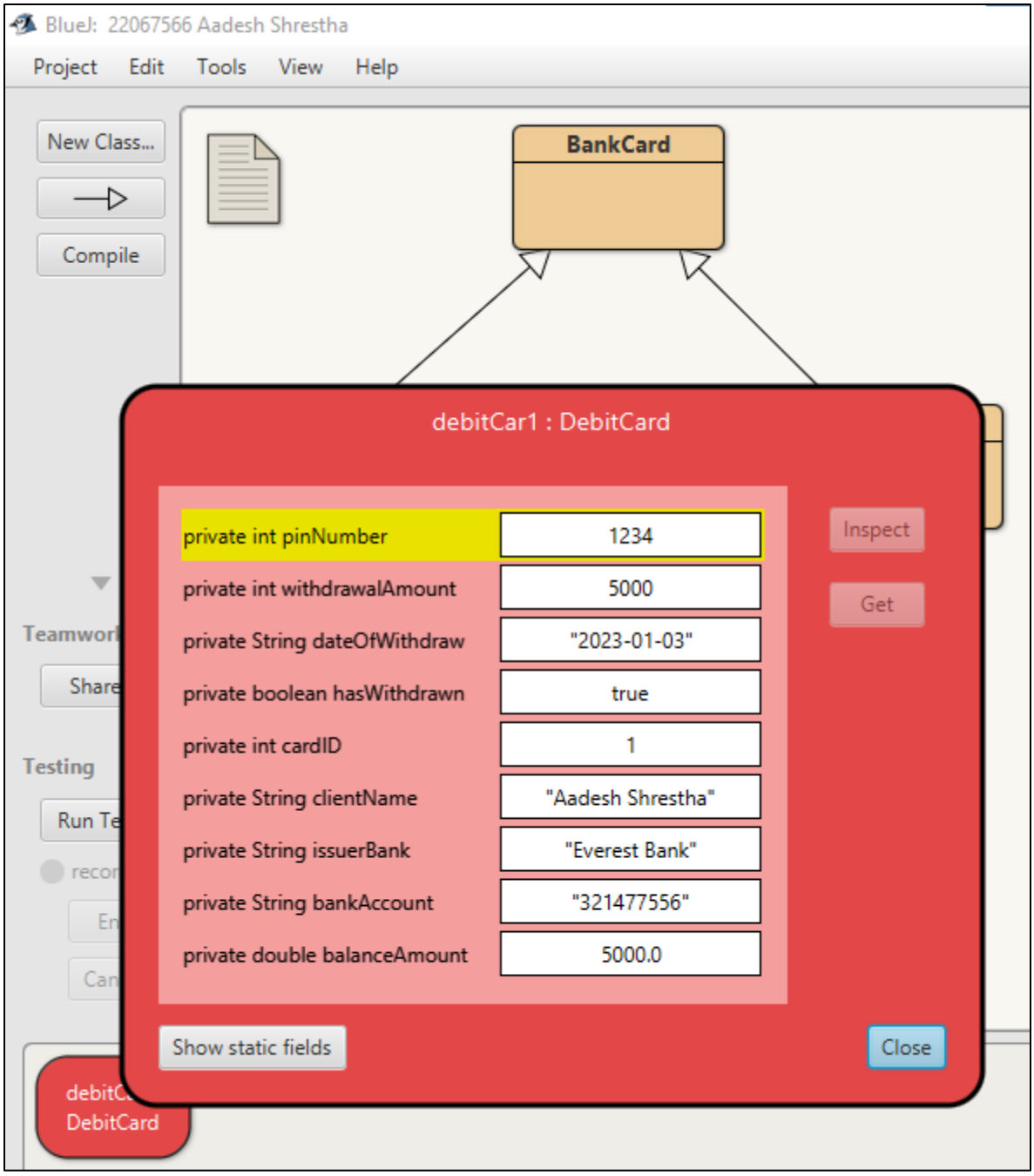



*Figure 8: Screenshot of re-inspection of the Debit Card class*

**5.2 Test 2:** Inspect Credit Card class, set the credit limit and reinspect the Credit Card class.

| | |
|---|---|
| Test No. | 2 |
| Objective | To inspect Credit Card class, set the credit limit and reinspect the Credit Card class. |
| Action | - The Credit Card is called with the following arguments:<br>cardID=1<br>issuerBank=”Everest Bank”<br>bankAccount= "321455776"<br>balanceAmount=20000<br>clientName=”Preeti Shrestha”<br>cvcNumber= 33227766<br>interestRate=12.5<br>expirationDate=”2024”<br>- Inspection of the Credit Card class<br>- void setCrediLimit() is called with the following arguments:<br>crediLimit=50000<br>gracePeriod=20<br>- Re-inspection of the Credit Card |
| Expected Result | A credit limit and grace period would be assigned for a credit card. |
| Actual Result | A credit limit and grace period was assigned. |
| Conclusion | The test is successful. |

*Table 2: To inspect Credit Card class, set the credit limit and reinspect the Credit Card class*

## Output Result:

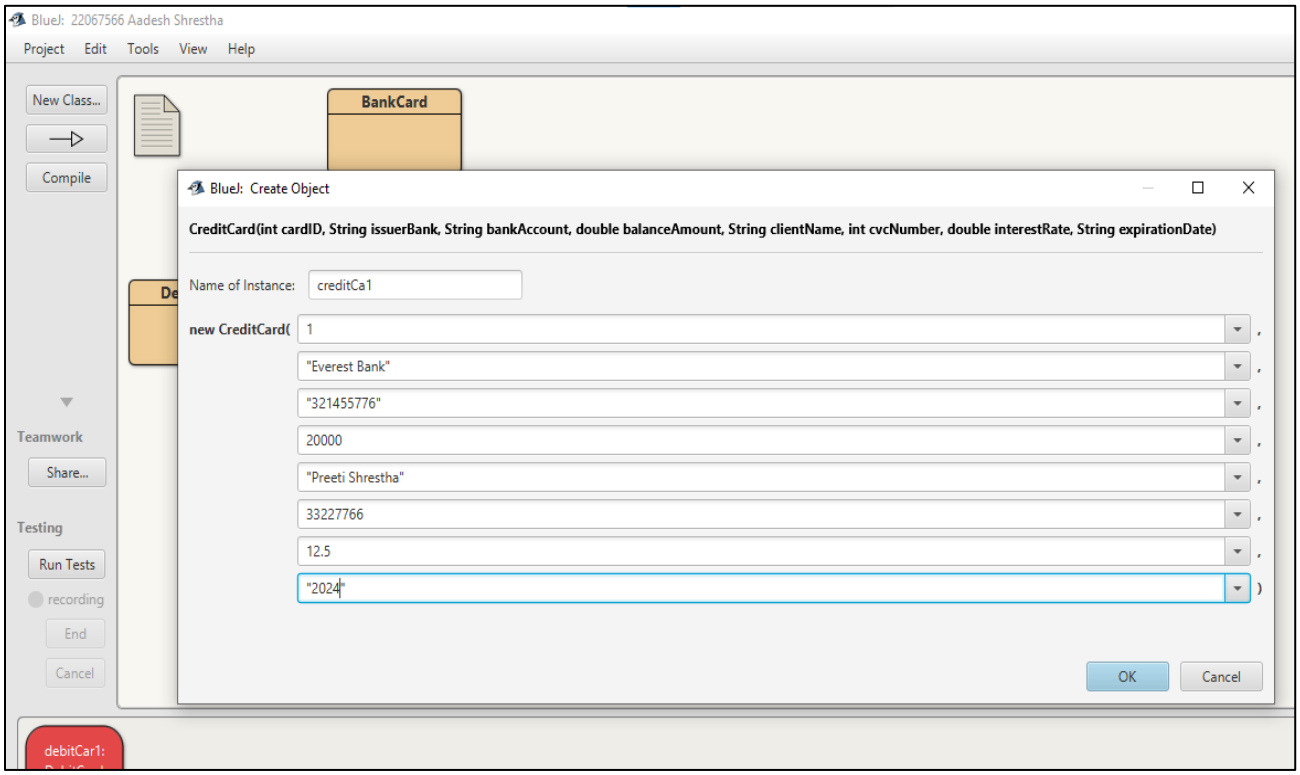



*Figure 9: Screenshot of assigning the data in the Credit Card class*

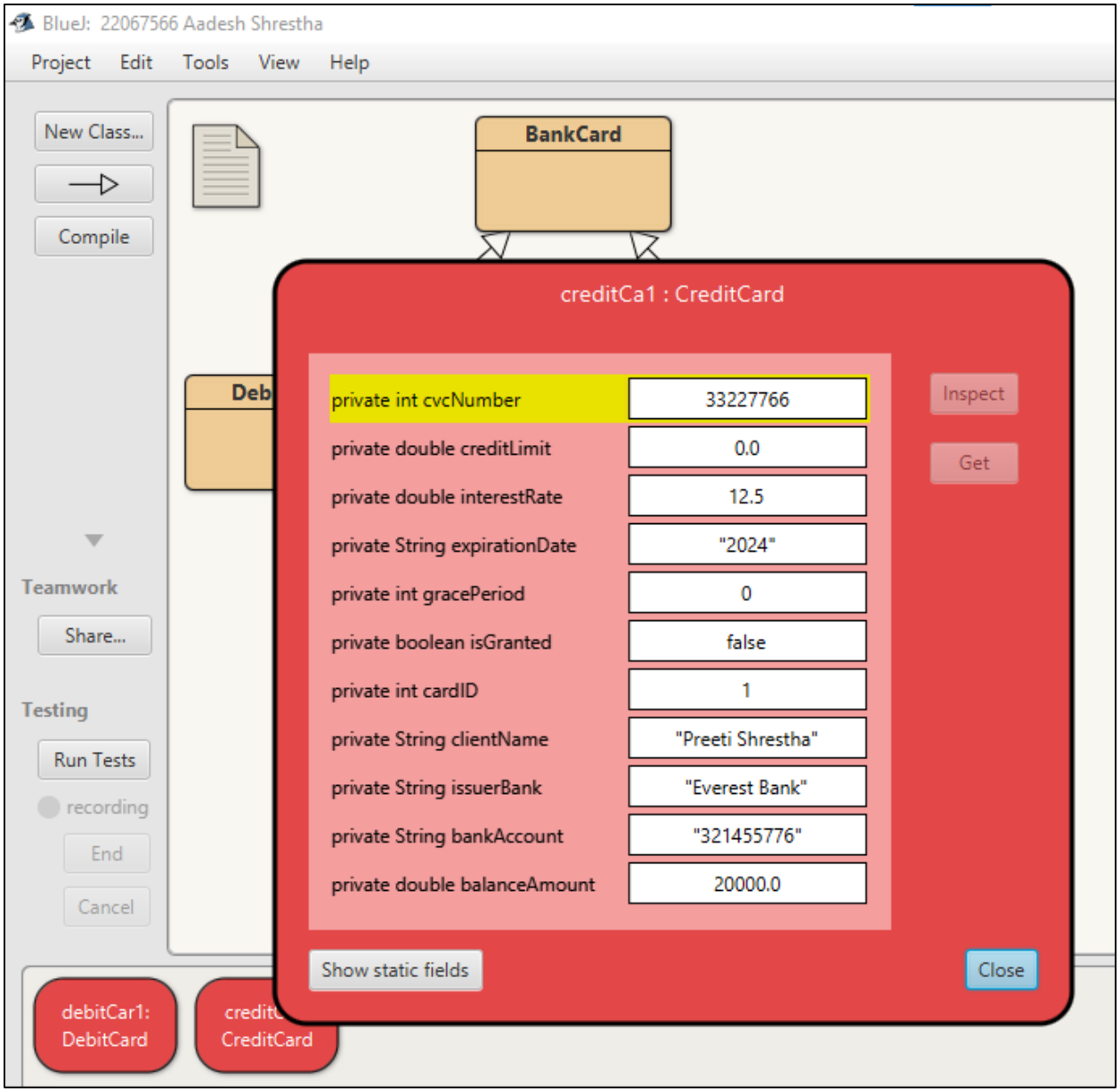



*Figure 10: Screenshot of the inspection of the Credit Card class*

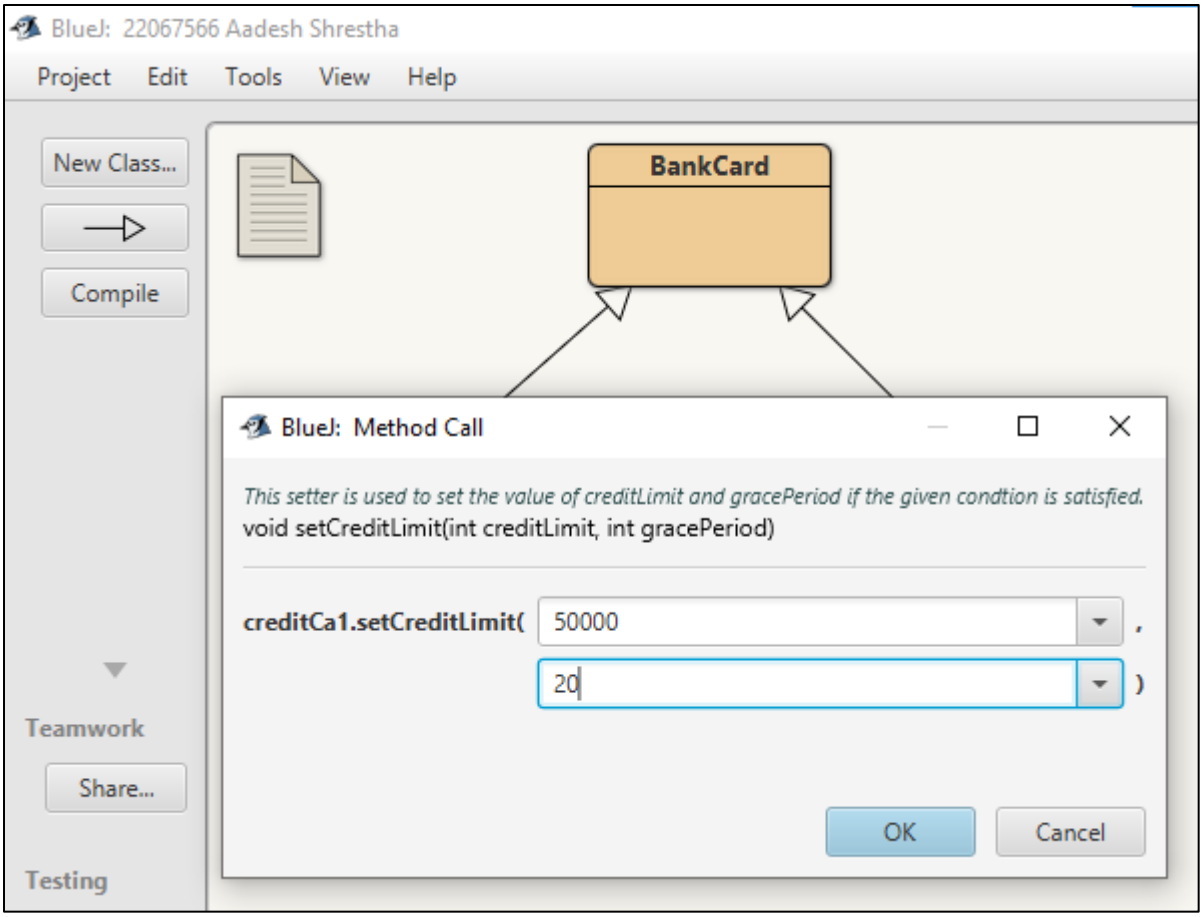



*Figure 11: Screenshot of assigning the data for void setCreditLimit*

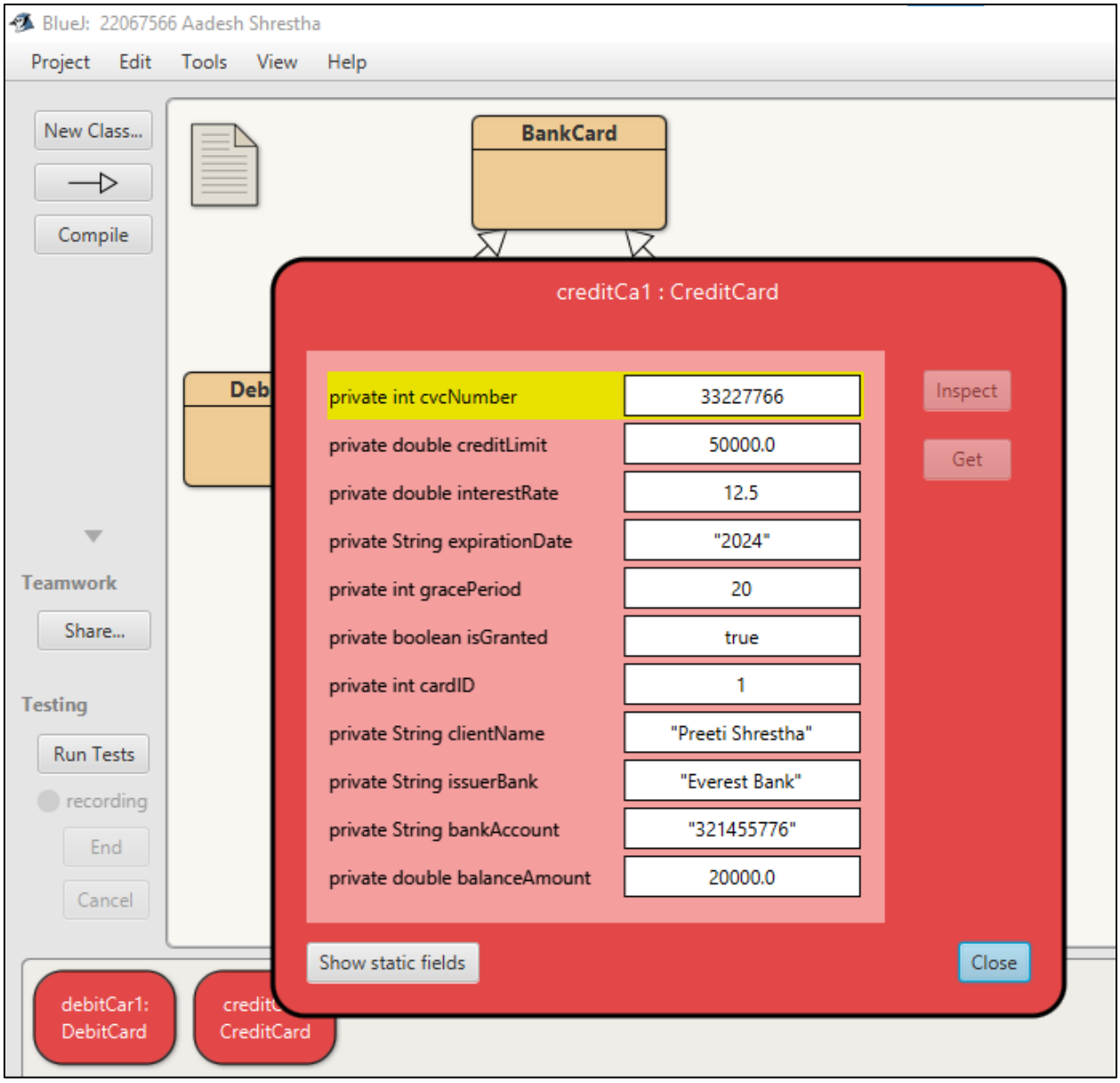



*Figure 12: Screenshot of re-inspection of the Credit Card class*

**5.3 Test 3:** Inspect Credit Card class again after cancelling the credit card.

| | |
|---|---|
| Test No. | 3 |
| Objective | To inspect Credit Card class again after cancelling the credit card. |
| Action | - Inspection of the Credit Card class<br>- void cancelCreditCard() is called with the following arguments:<br>cvcNumber=0<br>creditLimit=0<br>gracePeriod=0<br>-Re-inspection of the Credit Card |
| Expected Result | The credit card would be removed. |
| Actual Result | The credit card was removed. |
| Conclusion | The test is successful. |

*Table 3: To inspect Credit Card class again after cancelling the credit card*

**Output Result:**

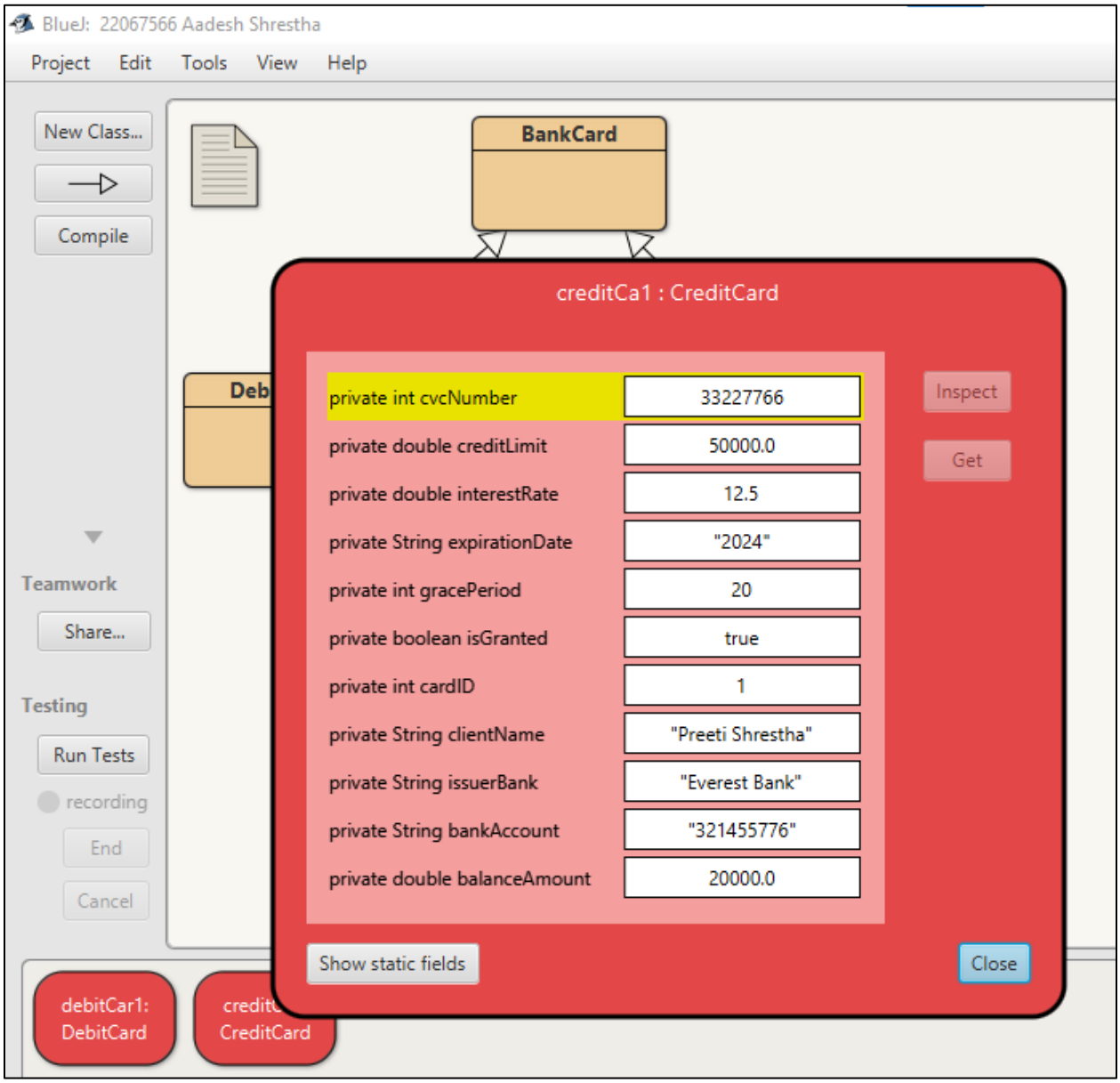



*Figure 13: Screenshot of inspection of the Credit Card class*

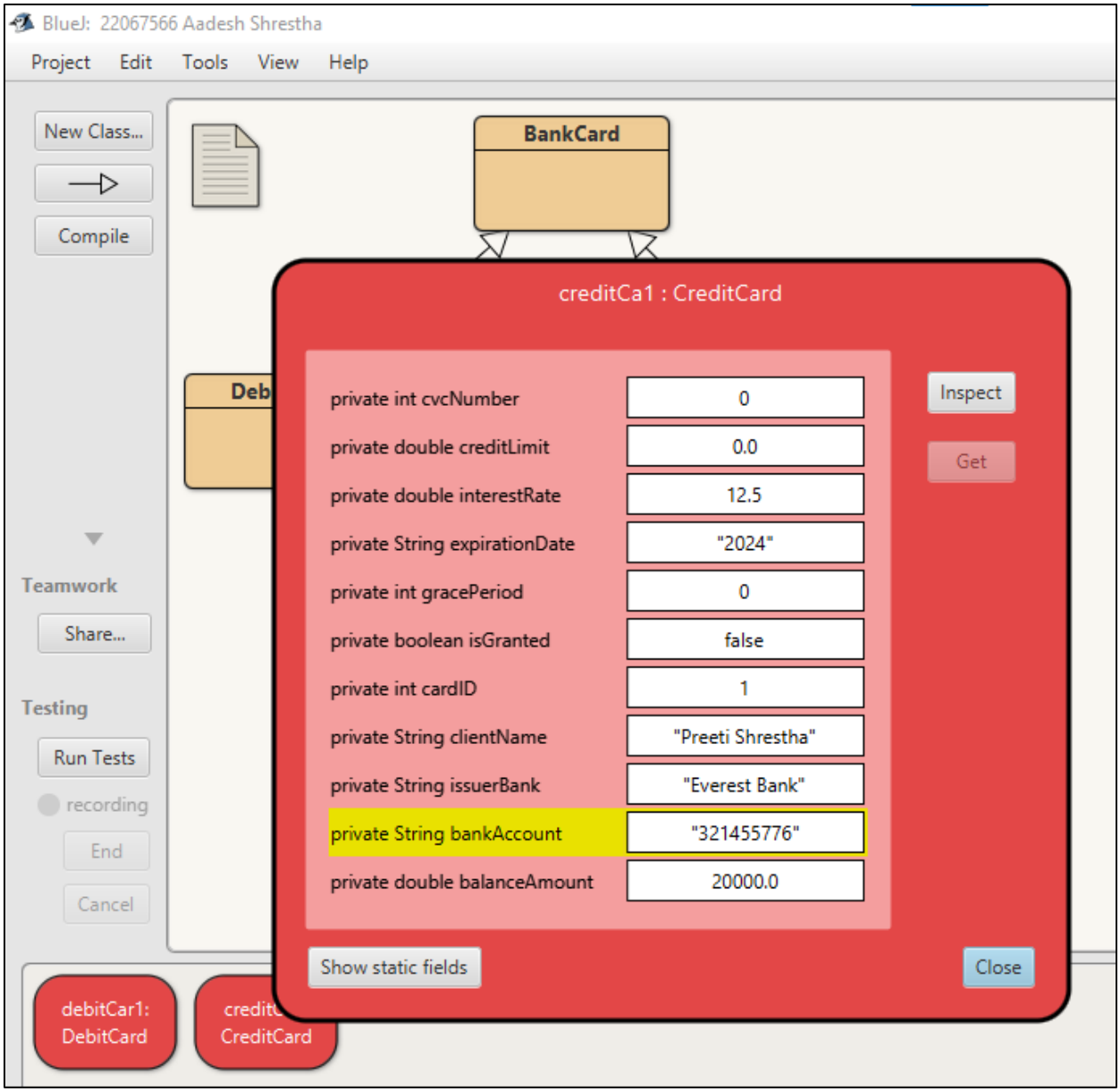



*Figure 14: Screenshot of re-inspection of the Credit Card class after calling cancelCreditCard method*

**5.4 Test 4:** Display the details of Debit Card and Credit Card classes.

| Test No. | 4 |
|---|---|
| Objective | To display the details of Debit Card and Credit Card classes. |
| Action | - void debitDisplay() is called<br>- void creditDisplay() is called |
| Expected Result | The content of debit card and credit card would be displayed. |
| Actual Result | The content of debit card and credit card was displayed. |
| Conclusion | The test is successful. |

*Table 4: To display the details of Debit Card and Credit Card classes*

**Output Result:**

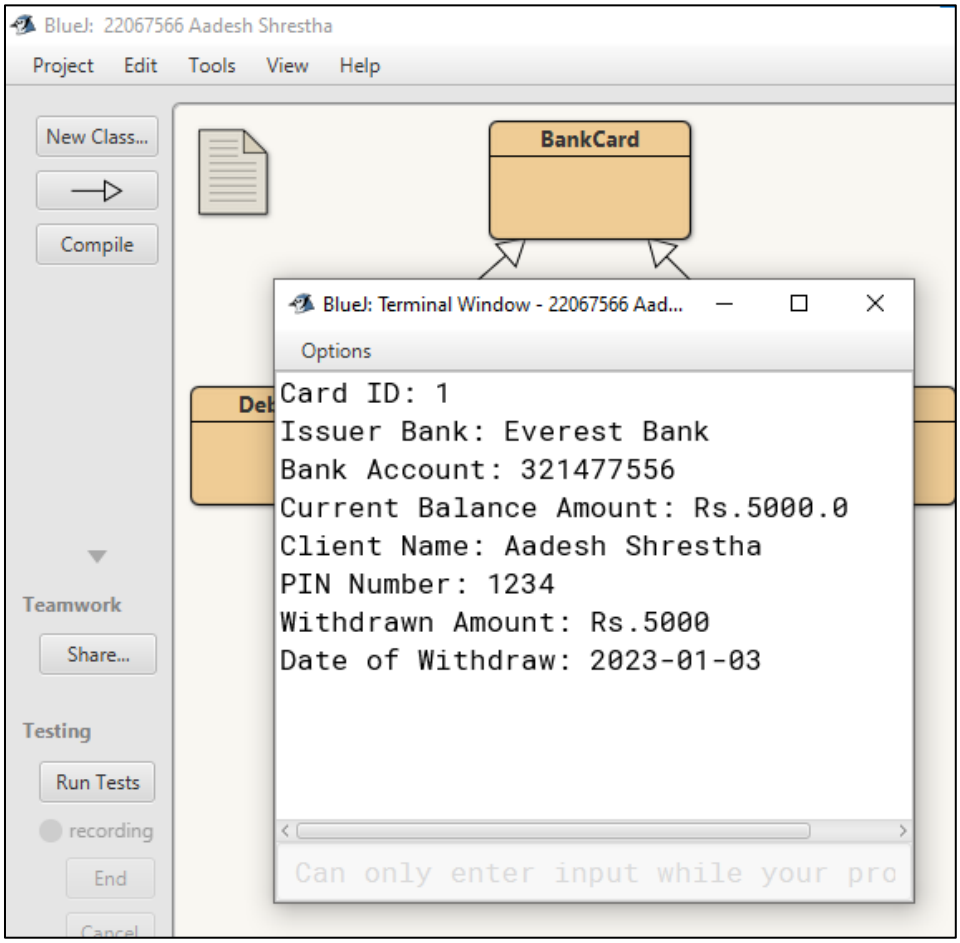



*Figure 15: Screenshot for displaying details of Debit Card class*

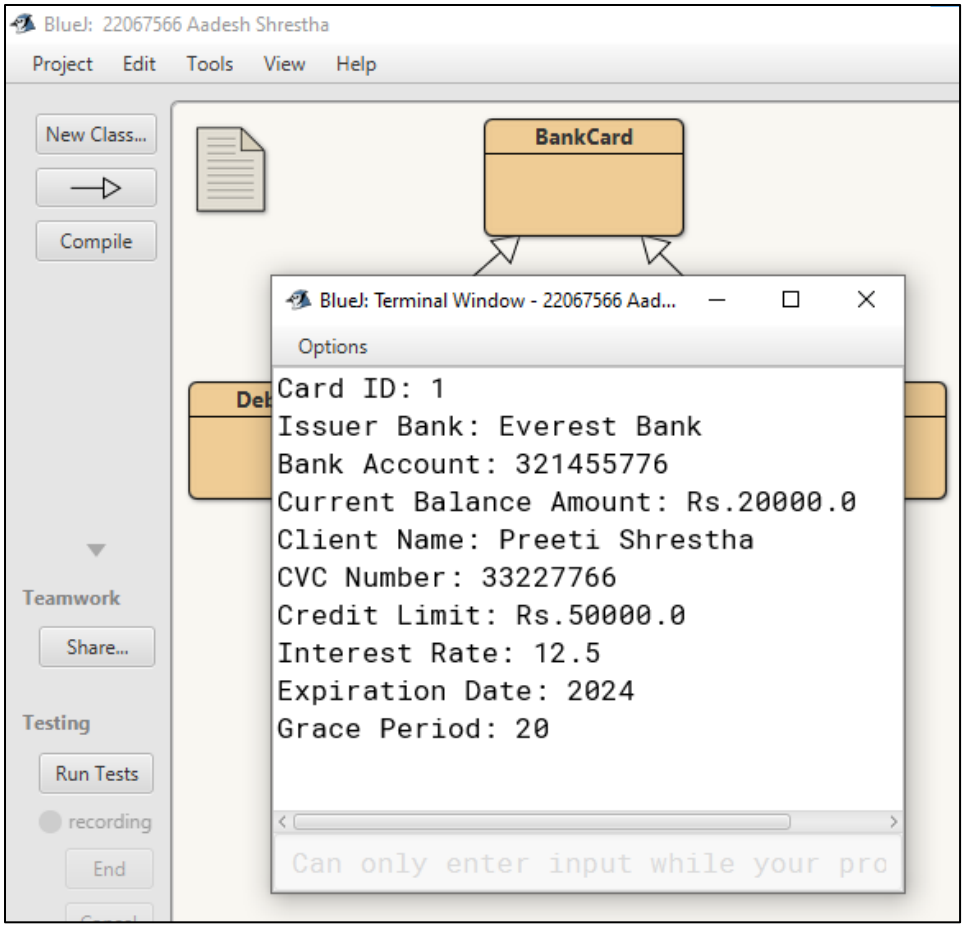



*Figure 16: Screenshot for displaying details of Credit Card class*

# 6. Error Detection and Correction

## 6.1 Error 1

```
if(clientName )!= ""){
    System.out.println("Client Name: " +this.clientName);
}
else{
    System.out.println("Client Name has not been registered.");
```

*Figure 17: Error 1*

This is a syntax error. This is caused when the format of the codes is incorrectly written. For instance, this error was caused by closing of the condition before it was written completely. This error was resolved by writing the code with valid expression.

```
if(clientName != ""){
    System.out.println("Client Name: " +this.clientName);
}
else{
    System.out.println("Client Name has not been registered.");
```

*Figure 18: Correction of error 1*

## 6.2 Error 2

```
public BankCard(int cardID, String issuerBank, String bankAccount, double balanceAmount)
{
    this.cardID = cardID;
    this.issuerBank = issuerBank;
    this.bankAccount = bankAccount;
    this.balanceAmount = balanceAmount;
    clientName = "";
}
```

*Figure 19: Error 2*

```
public DebitCard(String issuerBank, int cardID, String bankAccount, double balanceAmount, String clientName, int pinNumber)
{
   super(issuerBank, cardID, bankAccount, balanceAmount);
   super.setClientName(clientName);
   this.pinNumber = pinNumber;
   this.hasWithdrawn = false;
}
```

*Figure 20: Error 2*

It is a semantic error. This error is caused when a parameter of a constructor and super statement do not align with each other as they are of different data type and contain different values. This can be fixed by placing them in the right order.

```
public DebitCard(String issuerBank, int cardID, String bankAccount, double balanceAmount, String clientName, int pinNumber)
{
    super(cardID, issuerBank, bankAccount, balanceAmount);
    super.setClientName(clientName);
    this.pinNumber = pinNumber;
    this.hasWithdrawn = false;
}
```

*Figure 21: Correction of error 2*

## 6.3 Error 3

```
if (creditLimit <= super.getBalanceAmount() * 2.5){
    System.out.println("Your requested Amount exceeds the Credit Limit.");
}
else{
    this.creditLimit = creditLimit;
    this.gracePeriod = gracePeriod;
    this.isGranted = true;
}
```

*Figure 22: Error 3*

At first glance, there seems to be no error in the codes but if we look closely, we can see that the IF condition is executes wrong block of statements. There error is not spotted by the compiler as the codes are syntaxy correct but is logical flawed. Therefore, this is a logical error. This error causes the program to run completely inverse to itself.

```
if (creditLimit <= super.getBalanceAmount() * 2.5){
    this.creditLimit = creditLimit;
    this.gracePeriod = gracePeriod;
    this.isGranted = true;
}
else{
    System.out.println("Your requested Amount exceeds the Credit Limit.");
}
```

*Figure 23: Correction of error 3*

## 6.4 Types of Errors

1. Syntax Error

The most common error encountered by programmers while coding is a syntax error. While constructing a sentence, grammatic structure should be followed likewise while coding a similar structure should be followed which is called a syntax. These errors are also called Compile Time Error. These errors are detected immediately and a short description is provided to describe the error. (GeeksforGeeks, 2022)

2. Semantic Error

Semantic error is caused when Java statements are incorrectly written. Not all IDEs can detect this problem but BlueJ can detect them. Although these errors are detected, proper ways to resolve them are not given. (Calvanese, 2022)

3. Logical Error

Logical errors are difficult to spot as the syntax of the codes are correct but returns incorrect results. Compiler cannot detect these types of error. These errors occur when a programmer uses an incorrect idea or concept when writing the code. To put it simply, logical errors are errors that have an incorrect meaning. (GeeksforGeeks, 2022)

4. Run Time Error

Run Time error occur during the execution of the program or are detected when a user enters wrong inputs. Runtime errors happen when a program asks the computer to do an action that it is not capable of reliably performing. These error are also not detected by the compiler. (GeeksforGeeks, 2022)

## 7. Conclusion

In this project, I have gained a deeper understanding of object-oriented programming concepts in Java. I learn how to create classes, connect them and initialize values using constructors. I also learned about passing parameters through methods, concept of data encapsulation and how to use accessor and mutator methods to return and assigns data for a private attribute.

I encountered many challenges while writing this program, including syntax, semantic, and logical errors. Since this project posed unique set of problems, it was difficult to get a conclusive solution. Some of the errors were resolved by watching various informative videos in Youtube and asking my friends. But most of the problems were resolved by myself by using different ways to make the program work like juggling through different patterns of codes until I got it right.

The report for this project includes a testing section where the different classes and methods were tested, with screenshots of the output provided as evidence. The methods are described, detailing their purpose and their data. The program code is included in the appendix section, along with pseudo code and a class diagram for better understanding of the program.

Overall, I was able to grasp the core concepts used in Java and was able to successfully complete this coursework. I also improved my problem-solving skill as well as my researching skill.

## 8. References

BlueJ org, n.d. *About BlueJ.* [Online]
Available at: https://www.bluej.org/about.html
[Accessed 26 January 2023].

Calvanese, D., 2022. *Semantic errors.* [Online]
Available at: https://www.inf.unibz.it/~calvanese/teaching/06-07-ip/lecture-notes/uni10/node4.html
[Accessed 26 January 2023].

GeeksforGeeks, 2022. *Types of Errors in Java with Examples - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/types-of-errors-in-java-with-examples/
[Accessed 26 January 2023].

# 9. Appendix

## 9.1 Appendix for BankCard class

```
/**
 * Declaration of BankCard class
 * Stores value of cardID, clientName, issuerBank, bankAccount and balanceAmount
 * Display all the values if clientName is not empty else display all values except clientName
 */
public class BankCard
{
    //Attributes
    private int cardID;
    private String clientName;
    private String issuerBank;
    private String bankAccount;
    private double balanceAmount;

    //Constructor
    public BankCard(int cardID, String issuerBank, String bankAccount, double balanceAmount)
    {
        this.cardID = cardID;
        this.issuerBank = issuerBank;
```

```
        this.bankAccount = bankAccount;
        this.balanceAmount = balanceAmount;
        clientName = "";
    }

    //Mutator Method(Setter)
    public void setClientName(String clientName)
    {
        this.clientName = clientName;
    }

    public void setBalanceAmount(double balanceAmount)
    {
        this.balanceAmount = balanceAmount;
    }

    //Accessor method for cardID
    public int getCardID()
    {
        return this.cardID;
    }

    public String getClientName()
```

```
    {
        return this.clientName;
    }

    public String getIssuerBank()
    {
        return this.issuerBank;
    }

    public String getBankAccount()
    {
        return this.bankAccount;
    }

    public double getBalanceAmount()
    {
        return this.balanceAmount;
    }

    /**
     * display() is used to display the content of the bank card. It also checks if the client name is assign a value or not.
     */
```

```
    public void display()
    {
        System.out.println("Card ID: " +getCardID());
        System.out.println("Issuer Bank: " +getIssuerBank());
        System.out.println("Bank Account: " +getBankAccount());
        System.out.println("Current Balance Amount: Rs." +getBalanceAmount());

        //Checking if clientName is empty or not
        if(clientName != ""){
            System.out.println("Client Name: " +getClientName());
        }
        else{
            System.out.println("Client Name has not been registered.");
        }
    }
}
```

### 9.2 Appendix for DebitCard class

```
/**
 * Declaration of DebitCard class
 * Stores value of pinNumber, withdrawalAmount, dateOfWithdraw and hasWithdrawn
 * Takes out a withdrawal amount with values pinNumber, withdrawalAmount, dateOfWithdraw if the condtions
 * of pinNumber and withdrawalAmount are fulfilled.
 * Display all the values if hasWithdrawn is true else only display balanceAmount
 */
public class DebitCard extends BankCard
{
    //Attributes
    private int pinNumber;
    private int withdrawalAmount;
    private String dateOfWithdraw;
    private boolean hasWithdrawn;

    //Constructor
    public DebitCard(int cardID, String issuerBank, String bankAccount, double balanceAmount, String clientName, int pinNumber)
    {
        super(cardID, issuerBank, bankAccount, balanceAmount);
        super.setClientName(clientName);
```

```
        this.pinNumber = pinNumber;
        this.hasWithdrawn = false;
    }

    //Mutator Method(Setter)
    public void setWithdrawalAmount(int withdrawalAmount)
    {
        this.withdrawalAmount = withdrawalAmount;
    }

    //Accessor Method(Getter)
    public int getPinNumber()
    {
        return this.pinNumber;
    }

    public int getWithdrawalAmount()
    {
        return this.withdrawalAmount;
    }

    public String getDateOfWithdraw()
    {
```

```
        return this.dateOfWithdraw;
    }

    public boolean getHasWithdrawn()
    {
        return this.hasWithdrawn;
    }

    /**
     * withdraw() is used to withdraw money from the balanceAmount
     * withdraw is done if the pinNumber is valid and withdrawalAmount is less than balanceAmount
     * else suitable message is displayed
     */
    public void withdraw(int pinNumber, int withdrawalAmount, String dateOfWithdraw)
    {
        if (pinNumber == this.pinNumber){
            if(withdrawalAmount < super.getBalanceAmount() && withdrawalAmount > 0){
                super.setBalanceAmount(super.getBalanceAmount() - withdrawalAmount);
                this.pinNumber = pinNumber;
                this.withdrawalAmount = withdrawalAmount;
                this.dateOfWithdraw = dateOfWithdraw;
                hasWithdrawn = true;
```

```
            System.out.println("Your Transaction was Successful.");
        }
        else{
            System.out.println("You have Insufficent Balance.");
        }
    }
    else{
        System.out.println("The entered PIN number is incorrect.");
    }
}

/**
 * debitDisplay() is used to display the content of the debit card if the condition of hasWithdrawn is true.
 */
public void debitDisplay()
{
    if(hasWithdrawn == true){
        super.display();
        System.out.println("PIN Number: "+getPinNumber());
        System.out.println("Withdrawn Amount: Rs." +getWithdrawalAmount());
        System.out.println("Date of Withdraw: " +getDateOfWithdraw());
    }
```

```
        else{
            System.out.println("Current Balance Amount: Rs." +super.getBalanceAmount());
        }
    }
}
```

## 9.2 Appendix for CreditCard class

```
/**
 * Declaration of CreditCard class
 * Stores value of cvcNumber, creditLimit, interestRate, expirationDate, gracePeriod and isGranted
 * Sets creditLimit and gracePeriod with values creditLimit and gracePeriod if the condtions
 * of creditLimit is fulfilled
 * cancelCreditCard method cancels the credit card when called
 * Display all the values if isGranted is true else display all values except cvcNumber, creditLimit, garcePeriod
 */
public class CreditCard extends BankCard
{
    //Attributes
    private int cvcNumber;
    private double creditLimit;
    private double interestRate;
    private String expirationDate;
    private int gracePeriod;
    private boolean isGranted;

    //Constructor
```

```
    public CreditCard(int cardID, String issuerBank, String bankAccount, double balanceAmount, String clientName, int cvcNumber,
    double interestRate, String expirationDate)
    {
        super(cardID, issuerBank, bankAccount, balanceAmount);
        super.setClientName(clientName);
        this.cvcNumber = cvcNumber;
        this.interestRate = interestRate;
        this.expirationDate = expirationDate;
        this.isGranted = false;
    }

    //Mutator Method
    /**
     * This setter is used to set the value of creditLimit and gracePeriod if the given condtion is satisfied.
     */
    public void setCreditLimit(int creditLimit, int gracePeriod)
    {
        if (creditLimit <= super.getBalanceAmount() * 2.5 && creditLimit >= super.getBalanceAmount() * 2){
            this.creditLimit = creditLimit;
            this.gracePeriod = gracePeriod;
```

```
            this.isGranted = true;
        }
        else{
            System.out.println("Your requested Amount exceeds the Credit Limit.");
        }
    }

    //Accessor Method
    public int getCvcNumber()
    {
        return this.cvcNumber;
    }

    public double getCreditLimit()
    {
        return this.creditLimit;
    }

    public double getInterestRate()
    {
        return this.interestRate;
    }
```

```
    public String getExpirationDate()
    {
        return this.expirationDate;
    }

    public int getGracePeriod()
    {
        return this.gracePeriod;
    }

    public boolean getIsGranted()
    {
        return this.isGranted;
    }

    /**
     * cancelCreditCard() is used to remove the client's credit card.
     */
    public void cancelCreditCard()
    {
        this.cvcNumber = 0;
        this.creditLimit = 0;
        this.gracePeriod = 0;
```

```
        this.isGranted = false;
    }


    /**
     * creditDisplay() is used to display the content of the credit card if the condition of isGranted is true.
     */
    public void creditDisplay()
    {
        if (isGranted == true){
            super.display();
            System.out.println("CVC Number: " +getCvcNumber());
            System.out.println("Credit Limit: Rs." +getCreditLimit());
            System.out.println("Interest Rate: " +getInterestRate());
            System.out.println("Expiration Date: " +getExpirationDate());
            System.out.println("Grace Period: " +getGracePeriod());
        }
        else{
            super.display();
            System.out.println("CVC Number: " +getCvcNumber());
            System.out.println("Interest Rate: " +getInterestRate());
            System.out.println("Expiration Date: " +getExpirationDate());
        }
```

```
    }
}
```