



 slington college  
(इस्लिङ्टन कलेज)

## **CS4001NI Programming**

**30% Individual Coursework**

**2022-23 Autumn**

**Student Name: MIRAJ DEEP BHANDARI**

**London Met ID: 22067814**

**College ID: NP01CP4A220197**

**Group: L1C8**

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

**Assignment Submission Date: Thursday, January 26, 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 THE COURSEWORK.....	1
1.2 TOOLS USED.....	2
1.2.1 BLUEJ .....	2
1.2.2 Draw.IO .....	2
1.2.3 MS WORD .....	3
2. CLASS DIAGRAM.....	4
2.1 CLASS DIAGRAM OF BankCard .....	4
2.2 CLASS DIAGRAM OF DebitCard .....	5
2.3 CLASS DIAGRAM OF CreditCard .....	6
2.4 CLASS DIAGRAM OF BankCard, DebitCard and CreditCard .....	7
3. PSEUDOCODE.....	8
3.1 PSEUDOCODE OF CLASS BankCard.....	8
3.2 PSEUDOCODE OF CLASS DebitCard .....	13
3.3 PSEUDOCODE OF CLASS CreditCard .....	18
4. METHOD DESCRIPTIONS.....	24
4.1 METHOD DESCRIPTIONS OF CLASS BankCard.....	24
4.1.1 getCardId() .....	24
4.1.2 getClientName() .....	24
4.1.3 getIssuerBank() .....	24
4.1.4 getBankAccount().....	25
4.1.5 getBalanceAmount() .....	25
4.1.6 setClientName() .....	25
4.1.7 setBalanceAmount().....	25
4.1.8 display().....	26
4.2 METHOD DESCRIPTIONS OF CLASS DebitCard.....	26
4.2.1 getPinNumber() .....	26
4.2.2 getWithdrawalAmount().....	26

4.2.3 getDateofWithdrawal() .....	26
4.2.4 getHasWithdrawn() .....	27
4.2.5 setWithdrawalAmount() .....	27
4.2.6 withdraw().....	27
4.2.7 display().....	27
4.3 METHOD DESCRIPTIONS OF CLASS CreditCard .....	28
4.3.1 getCvcNumber() .....	28
4.3.2 getCreditLimit().....	28
4.3.3 getInterestRate() .....	28
4.3.4 getExpirationDate() .....	29
4.3.5 getGracePeriod().....	29
4.3.6 getIsGranted() .....	29
4.3.7 setCreditLimit() .....	29
4.3.8 CancelCreditCard() .....	30
4.3.9 display().....	30
5. Testing(Inspection) .....	31
5.1 Test 1 – To inspect the DebitCard class, withdraw the amount and re-inspect the Debit Card Class.....	31
5.2 Test 2 – To inspect the CreditCard class, Set the credit limit and re-inspect the CreditCard Class .....	34
5.3 Test 3 – To inspect the CreditCard class again after cancelling the credit card. .	37
5.4 Test 4 – To display the details of DebitCard and CreditCard classes. ....	42
6. ERROR DETECTION AND CORRECTION .....	56
6.1 FIRST ERROR AND ITS SOLUTION. ....	58
6.2 SECOND ERROR AND ITS SOLUTION. ....	60
6.3 THIRD ERROR AND ITS SOLUTION. ....	62
6.4 FOUR ERROR AND ITS SOLUTION. ....	64
7. CONCLUSION .....	66
8. APPENDIX .....	69
8.1 CODE OF BankCard.java.....	69

8.2 CODE OF DebitCard.java.....	72
8.3 CODE OF CreditCard.java .....	77
9. References.....	83

## Table of Figures

Figure 1: Class diagram of BankCard .....	4
Figure 2: Class diagram of DebitCard .....	5
Figure 3: Class diagram of CreditCard .....	6
Figure 4: Class diagram of BankCard, DebitCard and CreditCard .....	7
Figure 5: Screenshot of assigning the data in DebitCard class .....	32
Figure 6: Screenshot for the inspection of DebitCard class.....	32
Figure 7: Screenshot of re-inspection of DebitCard class .....	33
Figure 8: Sceenshot of giving arguments to the withdraw method .....	33
Figure 9: Screenshot of assigning the data in CreditCard class .....	35
Figure 10: Screenshot for the inspection of CreditCard class .....	35
Figure 11:Screenshot of giving arguments to the setCreditLimit method .....	36
Figure 12: Screenshot of re-inspection of CreditCard class .....	36
Figure 13: Screenshot of assigning the data in CreditCard class.....	38
Figure 14: Screenshot of giving arguments to the setCreditLimit method .....	38
Figure 15:Screenshot of inspection of creditcard before cancelling credit card.....	39
Figure 16:Screenshot of cancelling Credit Card .....	40
Figure 17:Screenshot of re-inspection of CreditCard class after cancelling credit class	41
Figure 18: Screenshot of assigning the data in DebitCard class .....	45
Figure 19:Screenshot of calling display method before withdraw.....	46
Figure 20: Screenshot of output of display method before withdraw .....	47
Figure 21: Screenshot of giving values to the withdraw method .....	47
Figure 22: Screenshot of calling display method after withdraw.....	48
Figure 23: Screenshot of output of display method after withdraw .....	49

Figure 24: Screenshot of assigning the data in CreditCard class.....	50
Figure 25: Screenshot of calling display method before setting credit limit .....	51
Figure 26: Screenshot of output of display method before setting credit limit .....	52
Figure 27: Screenshot of giving values to setCreditLimit.....	53
Figure 28: Screenshot of calling display method after setting credit limit .....	54
Figure 29: Screenshot of output of display method after setting credit limit .....	55
Figure 32: First Error (logical error) .....	58
Figure 33: Solution of the first error (logical error) .....	59
Figure 34: Second Error (Syntax error) .....	60
Figure 35: Solution of the second error (Syntax error) .....	61
Figure 36: Third Error (Syntax error) .....	62
Figure 37: Solution of the Third Error (Syntax error) .....	63
Figure 38: Fourth Error (Semantic error) .....	64
Figure 39: Solution of the fourth error (Semantic error).....	65

## Table of Tables

Table 1: To inspect the DebitCard class, withdraw the amount and re-inspect the Debit Card Class .....	31
Table 2: To inspect the CreditCard class, Set the credit limit and re-inspect the CreditCard Class .....	34
Table 3: To inspect the CreditCard class again after cancelling the credit card. ....	37
Table 4: To display the details of DebitCard and CreditCard classes. ....	44

# 1. INTRODUCTION

## 1.1 ABOUT THE COURSEWORK

Java is a popular, general-purpose programming language that is designed to minimize implementation dependencies and is based on the concepts of classes and objects. It is a versatile computing platform that is widely used for application development, and is known for its speed, security, and reliability. It is widely used to develop various applications such as laptops, data centers, game consoles, scientific supercomputers, cell phones, etc (Hartman, 2022).

The purpose of this coursework is to conduct a thorough examination of the functionality of three Java classes, namely BankCard, DebitCard, and CreditCard, and their significance in modeling the functionality of bank credit and debit cards. These classes have been designed to emulate the functionality of bank credit and debit cards in a programmatic context. The BankCard class serves as the foundation for these classes, and comprises the essential properties of a credit card or debit card, such as cardId, clientName, issuerBank, bankAccount, and balanceAmount. Additionally, it features a constructor method that enables the creation and initialization of objects with these properties, facilitating the implementation of these functionalities in various applications.

The DebitCard class extends the functionality of the BankCard class, by incorporating additional properties and functionality specific to debit cards, such as pinNumber, withdrawalAmount, dateofWithdrawal and hasWithdrawn. It also includes a constructor method that facilitates the creation and initialization of objects with these properties, as well as a withdraw method that enables withdrawal of funds from the debit card.

Similarly, the CreditCard class extends the functionality of the BankCard class, by incorporating additional properties and functionality specific to credit cards, such as cvcNumber, creditLimit, interestRate, expirationDate, gracePeriod, and isGranted. It also includes a constructor method that facilitates the creation and initialization of objects with these properties, as well as a setCreditLimit method that sets the credit limit of the credit card and checks the balance of the credit card.

In summary, the primary objective of these classes is to model the functionality of bank credit cards and debit cards in a programmatic context, thus allowing for easy manipulation and utilization of these objects in a program. By defining these classes, we can create objects that emulate real-world credit cards and debit cards, and utilize them to perform tasks such as checking the balance of a card, withdrawing funds from a debit card, setting the credit limit of a credit card, among others."

## **1.2 TOOLS USED**

### **1.2.1 BLUEJ**

BlueJ is a free Java development platform created by Monash University for teaching object-oriented programming. It requires JDK 1.3 or later and has a user-friendly interface with tools specific to education, as well as standard development tools. It offers convenient features like sample programs and easy debugging and can run on multiple platforms. It also allows for interaction with objects (harleenk\_99 & mitalibhola94, 2022)

I used BlueJ as my main development platform for my coursework and found it to be extremely helpful in identifying and resolving errors in my code. It also provided a wide range of features and options for Java programming, including support for object-oriented programming principles and easy implementation of these principles. Overall, BlueJ proved to be an effective tool for my programming needs

### **1.2.2 Draw.IO**

Draw.io is a user-friendly and versatile diagram software with a range of features for creating professional diagrams and charts. It has a drag-and-drop interface and an automatic layout function for arranging elements. It also offers a variety of shapes and



visual elements for creating unique diagrams that can be used in different fields like software development, data visualization, project management and more (Hope, 2020).

Draw.io was a key tool in my coursework, allowing me to easily create class diagrams for BankCard, DebitCard, and CreditCard. It has a user-friendly drag-and-drop interface, and offers pre-built shapes which streamlined the process of creating diagrams. Overall, it was crucial for creating class diagrams efficiently and effectively.

### **1.2.3 MS WORD**

Microsoft Word, also known as Winword, MS Word, or simply Word, is a widely-used word processing application developed by Microsoft and part of the Microsoft Office Suite. It is a versatile tool designed to streamline document creation, editing, and management and widely used by professionals and students for various tasks such as reports and presentations (Hope, 2020).

I used Microsoft Word, also known as MS Word or Word, to create report documents in my coursework. It is a widely-used and user-friendly word processor with features such as image, shape, icon, graph and chart insertions. It proved to be an essential tool for my coursework by allowing me to create professional-looking report documents efficiently.

## 2. CLASS DIAGRAM

### 2.1 CLASS DIAGRAM OF BankCard



Figure 1: Class diagram of BankCard

## 2.2 CLASS DIAGRAM OF DebitCard

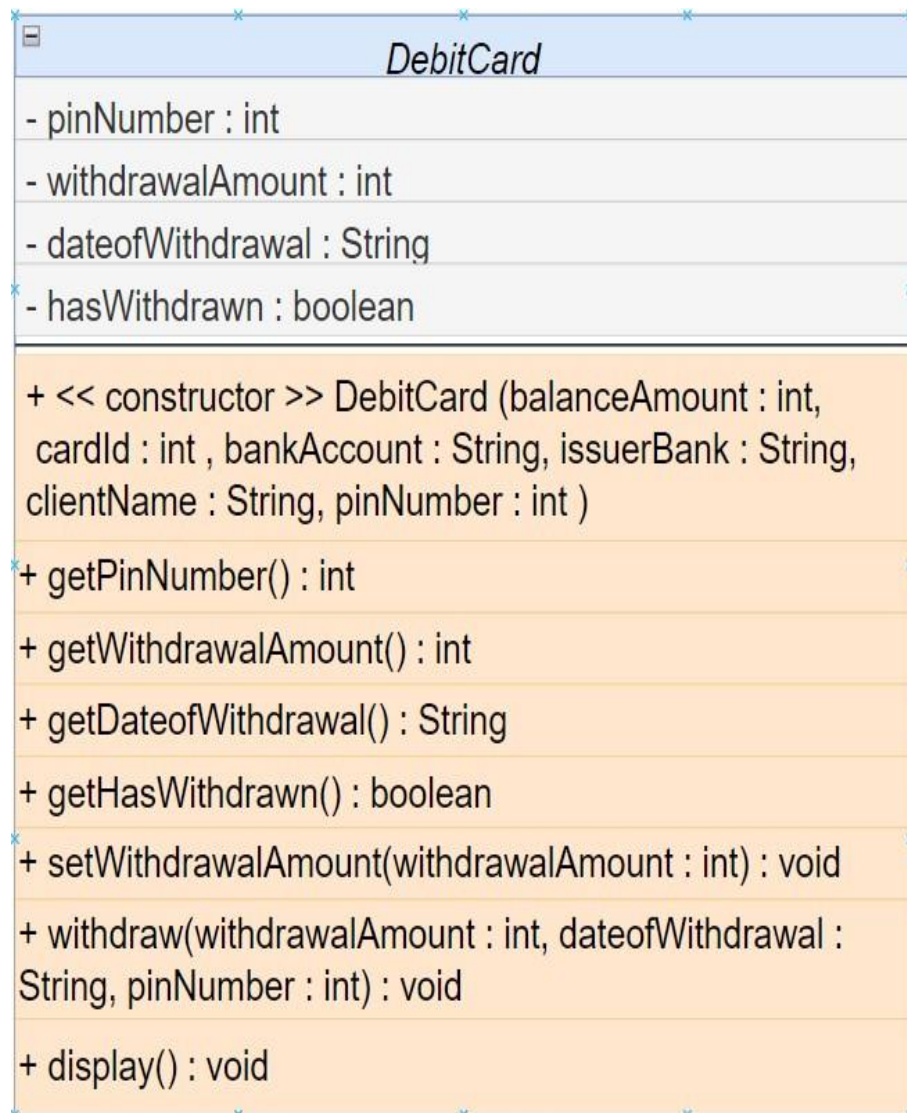


Figure 2: Class diagram of DebitCard

## 2.3 CLASS DIAGRAM OF CreditCard



Figure 3: Class diagram of CreditCard

## 2.4 CLASS DIAGRAM OF BankCard, DebitCard and CreditCard

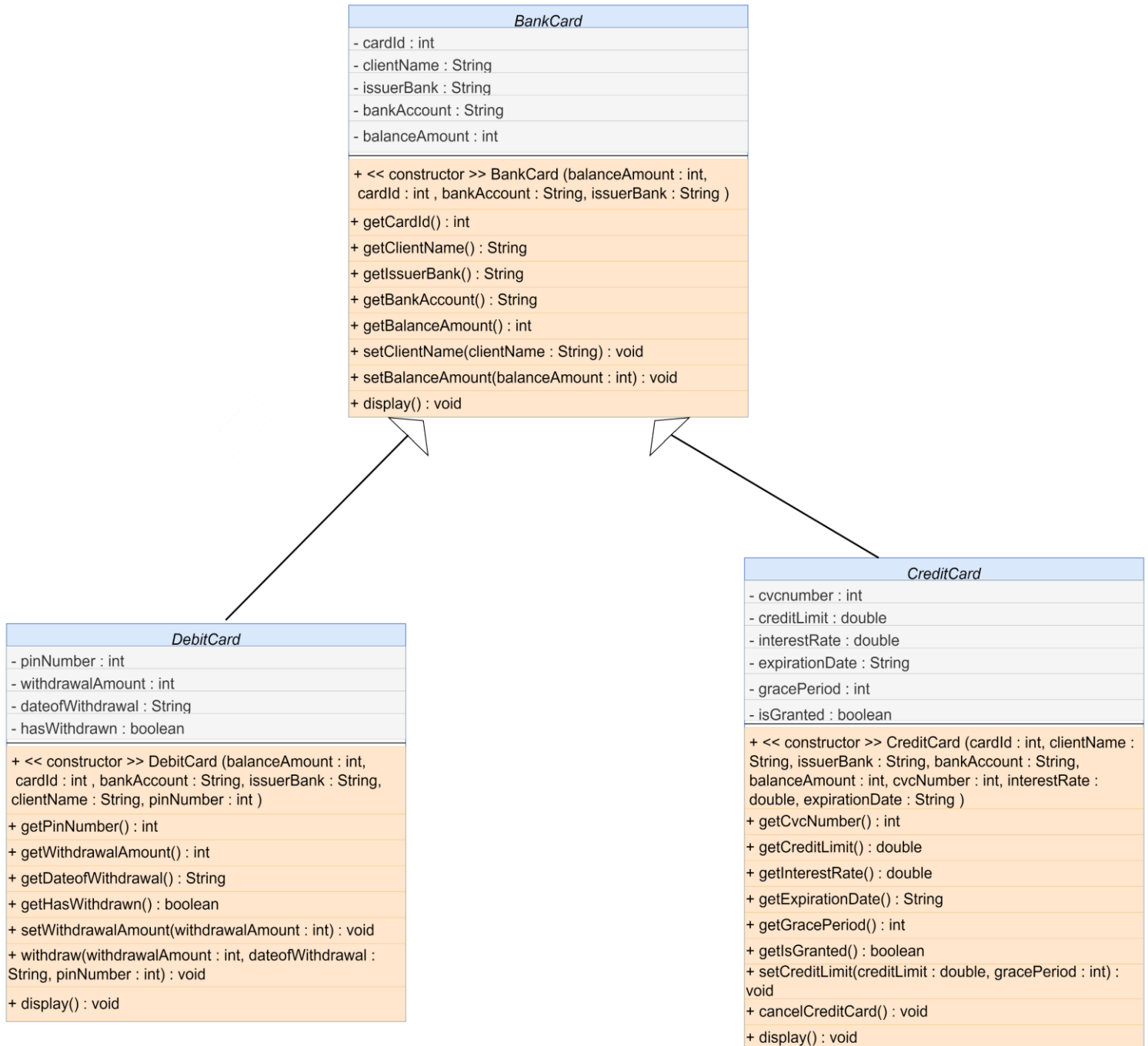


Figure 4: Class diagram of BankCard, DebitCard and CreditCard

### 3. PSEUDOCODE

#### 3.1 PSEUDOCODE OF CLASS BankCard

**CREATE** a parent class **BankCard**

**DO**

**DECLARE** an instance variable **cardId** as Integer

**DECLARE** an instance variable **issuerBank** as String

**DECLARE** an instance variable **bankAccount** as String

**DECLARE** an instance variable **balanceAmount** as integer

**DECLARE** an instance variable **clientName** as String

**CREATE** a constructor **BankCard** with parameters **balanceAmount**, **cardId**, **bankAccount**, **issuerBank**

**DO**

**ASSIGN** instance variable **balanceAmount** with parameter variable **balanceAmount**

**ASSIGN** instance variable **cardId** with parameter variable **cardId**

**ASSIGN** instance variable **bankAccount** with parameter variable **bankAccount**

**ASSIGN** instance variable **issuerBank** with parameter variable **issuerBank**

**ASSIGN** instance variable **clientName** to **empty**

**END DO**

**CREATE** a getter method **getCardId()** with return type integer

**DO**

**RETURN** instance variable **cardId**

**END DO**

**CREATE** a getter method **getIssuerBank()** with return type String

**DO**

**RETURN** instance variable **issuerBank**

**END DO**

**CREATE** a getter method **getBankAccount()** with return type String

**DO**

**RETURN** instance variable **bankAccount**

**END DO**

**CREATE** a getter method **getBalanceAmount()** with return type integer

**DO**

**RETURN** instance variable **balanceAmount**

**END DO**

**CREATE** a getter method **getClientName()** with return type String

**DO**

**RETURN** instance variable **clientName**

**END DO**



**CREATE** a setter method **setClientName()** which have no return type with parameter **clientName**

**DO**

**ASSIGN** instance variable **clientName** with parameter variable **clientName**

**END DO**

**CREATE** a setter method **setBalanceAmount()** which have no return type with parameter **balanceAmount**

**DO**

**ASSIGN** instance variable **balanceAmount** with parameter variable **balanceAmount**

**END DO**

**CREATE** a **display()** method which have no return type

**DO**

**IF** **clientName** is **Empty**

```
    PRINT "The client is not assigned"

    PRINT " The value of cardId "

    PRINT " The value of issuerBank "

    PRINT " The value of bankAccount "

    PRINT " The value of balanceAmount "

END IF

ELSE

    PRINT "The value of clientName"

    PRINT "The value of cardId"

    PRINT "The value of issuerBank"

    PRINT "The value of bankAccount"

    PRINT "The value of balanceAmount"

END DO

END DO
```

### 3.2 PSEUDOCODE OF CLASS DebitCard

**CREATE** a class **DebitCard** which is a child of **BankCard**

**DO**

**DECLARE** an instance variable **pinNumber** as integer

**DECLARE** an instance variable **withdrawalAmount** as integer

**DECLARE** an instance variable **dateofWithdrawal** as String

**DECLARE** an instance variable **hasWithdrawn** as Boolean

**CREATE** a constructor **DebitCard** which have parameters **balanceAmount**, **cardId**, **bankAccount**, **issuerBank**, **clientName**, **pinNumber**

**DO**

**CALL** a parent class constructor with parameters **balanceAmount**, **cardId**, **bankAccount** and **issuerBank**

**CALL** a setter method **setClientName** for attribute **clientName**

**ASSIGN** instance variable **pinNumber** with parameter variable **pinNumber**

**SET** instance variable **hasWithdrawn** to **false**

**END DO**

**CREATE** a getter method **getPinNumber()** with return type integer

**DO**

**RETURN** instance variable **pinNumber**

**END DO**

**CREATE** a getter method **getWithdrawalAmount()** with return type integer

**DO**

**RETURN** instance variable **withdrawalAmount**

**END DO**

**CREATE** a getter method **getDateofWithdrawal()** with return type String

**DO**

**RETURN** instance variable **dateofWithdrawal**

**END DO**

**CREATE** a getter method **getHasWithdrawn()** with return type boolean

**DO**

**RETURN** instance variable **hasWithdrawn**

**END DO**

**CREATE** a setter method **setWithdrawalAmount()** which have no return type which have parameter **withdrawalAmount**

**DO**

**ASSIGN** instance variable **withdrawalAmount** with parameter variable **withdrawalAmount**

**END DO**

**CREATE withdraw()** method which have no return type with parameters **withdrawalAmount**, **dateofWithdrawal**, **pinNumber**

**DO**

**IF** instance variable **pinNumber** is equal to parameter variable **pinNumber**

**IF** **withdrawalAmount** is lesser equal to **balanceAmount**

**CALL** a setter method **setBalanceAmount()** which computes **balanceAmount** and **withdrawalAmount**

**ASSIGN** instance variable **withdrawalAmount** with parameter variable **withdrawalAmount**

**ASSIGN** instance variable **dateofWithdrawal** with parameter variable **dateofWithdrawal**

**SET** instance variable **hasWithdrawn** to **True**

**PRINT** "The value of newBalance"

**END IF**

**ELSE**

**PRINT** "Insufficient funds. You cannot withdraw"

**END IF**

**ELSE**

**PRINT** "The pin is invalid please! Enter correct pin"

**END DO**

**CREATE display()** method which have no return type

**DO**

**IF** hasWithdrawn is equal to **true**

**CALL** a **parent** class **display()** method

**PRINT** "The value of pinNumber"

**PRINT** "The value of withdrawalAmount"

**PRINT** "The value of dateofWithdrawal"

**END IF**

**ELSE**

**PRINT** "The value of balanceAmount"

**END DO**

**END DO**

### 3.3 PSEUDOCODE OF CLASS CreditCard

**CREATE** a class **CreditCard** which is a child of **BankCard**

**DO**

**DECLARE** an instance variable **cvcNumber** as integer

**DECLARE** an instance variable **creditLimit** as double

**DECLARE** an instance variable **interestRate** as double

**DECLARE** an instance variable **expirationDate** as String

**DECLARE** an instance variable **gracePeriod** as integer

**DECLARE** an instance variable **isGranted** as boolean



**CREATE** a constructor **CreditCard** which have parameters **cardId**, **clientName**, **issuerBank**, **bankAccount**, **balanceAmount**, **cvcNumber**, **interestRate**, **expirationDate**

**DO**

**CALL** a **parent** class constructor with parameters **balanceAmount**, **cardId**, **bankAccount** and **issuerBank**

**CALL** a setter method **setClientName()** for attribute **clientName**

**ASSIGN** instance variable **cvcNumber** with parameter variable **cvcNumber**

**ASSIGN** instance variable **interestRate** with parameter variable **interestRate**

**ASSIGN** instance variable **expirationDate** with parameter variable **expirationDate**

**SET** instance variable **isGranted** to **false**

**END DO**

**CREATE** a getter method **getCvcNumber()** with return type integer

**DO**

**RETURN** instance variable **cvcNumber**

**END DO**

**CREATE** a getter method **getCreditLimit()** with return type double

**DO**

**RETURN** instance variable **creditLimit**

**END DO**

**CREATE** a getter method **getInterestRate()** with return type double

**DO**

**RETURN** instance variable **interestRate**

**END DO**

**CREATE** a getter method **getExpirationDate()** with return type String

**DO**

**RETURN** instance variable **expirationDate**

**END DO**

**CREATE** a getter method **getGracePeriod()** with return type integer

**DO**

**RETURN** instance variable **gracePeriod**

**END DO**

**CREATE** a getter method **getIsGranted()** with return type boolean

**DO**

**RETURN** instance variable **isGranted**

**END DO**

**CREATE** a setter method **setCreditLimit()** which have no return type which have parameters **creditLimit** and **gracePeriod**

**DO**

**IF** **creditLimit** is lesser equal to **two point five** times **balanceAmount**

**ASSIGN** instance variable **creditLimit** with parameter variable **creditLimit**

**ASSIGN** instance variable **gracePeriod** with parameter variable **gracePeriod**

**SET** instance variable **isGranted** to **True**

**PRINT** “ the value of credit limit and grace period ”

**END IF**

**ELSE**

**Print** "you cannot grant credit"

**END DO**

**CREATE** a method **cancelCreditCard()** which have no return type

**DO**

**SET** instance variable **cvcNumber** to **zero**

**SET** instance variable **creditLimit** to **zero**

**SET** instance variable **gracePeriod** to **zero**

**SET** instance variable **isGranted** to **False**

**END DO**

**CREATE** a method **display()** which have no return type

**DO**

**CALL** a parent class **display()** method

**IF isGranted is equal to true**

**PRINT** “the value of cvcNumber”

**PRINT** “the value of creditLimit”

**PRINT** “the value of gracePeriod”

**PRINT** “the value of interestRate”

**PRINT** “the value of expirationDate”

**END IF**

**ELSE**

**PRINT** “the value of cvcNumber”

**PRINT** “the value of interestRate”

**PRINT** “the value of expirationDate”

**END DO**

**END DO**

## 4. METHOD DESCRIPTIONS

### 4.1 METHOD DESCRIPTIONS OF CLASS BankCard

#### 4.1.1 getCardId()

"**getCardId()**" method of class BankCard returns the **cardId property** using the "**return**" statement and allows user to access it from outside the class, it's a **publicly** accessible method with **integer** return type.

#### 4.1.2 getClientName()

"**getClientName()**" method of class BankCard returns the **clientName property** using the "**return**" statement, it allows user to access it from outside the class, it's a **publicly** accessible method with **String** return type.

#### 4.1.3 getIssuerBank()

"**getIssuerBank()**" method of class BankCard returns the **issuerBank property** using the "**return**" statement, it allows user to access it from outside the class, it's a **publicly** accessible method with **String** return type.

#### 4.1.4 getBankAccount()

"**getBankAccount()**" method of class BankCard returns the **bankAccount** property using the "**return**" statement, it allows user to access it from outside the class, it's a **publicly** accessible method with **String** return type.

#### 4.1.5 getBalanceAmount()

"**getBalanceAmount**" method of class BankCard returns the **balanceAmount** property using the "**return**" statement, it allows user to access it from outside the class, it's a **publicly** accessible method with **integer** return type.

#### 4.1.6 setClientName()

"**setClientName**" of BankCard class sets the private property "**clientName**" with passed parameter, allows encapsulation and protected modification, **publicly** accessible and with **no return type**. Takes in single "**clientName**" parameter of type "**String**".

#### 4.1.7 setBalanceAmount()

"**setBalanceAmount()**" of BankCard class sets the private property "**balanceAmount**" with passed parameter, allows encapsulation and protected modification, **publicly** accessible and with **no return type**. Takes in single "**balanceAmount**" parameter of type "**int**".

#### 4.1.8 display()

"**display()**" method of BankCard class, **publicly** accessible, **no parameter** method, it **prints** the current state of class properties such as "clientName", "cardId", "issuerBank", "bankAccount" and "balanceAmount".

### 4.2 METHOD DESCRIPTIONS OF CLASS DebitCard

#### 4.2.1 getPinNumber()

"**getPinNumber()**" of DebitCard class returns the "**pinNumber**" property using the "**return**" statement, allows user to access it from outside the class, it's a publicly accessible method with **integer** return type

#### 4.2.2 getWithdrawalAmount()

"**getWithdrawalAmount()**" of DebitCard class returns the "**withdrawalAmount**" property using the "**return**" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **integer** return type.

#### 4.2.3 getDateofWithdrawal()

"**getDateofWithdrawal()**" of DebitCard class returns the "**dateofWithdrawal**" property using the "**return**" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **String** return type.



#### 4.2.4 getHasWithdrawn()

"**getHasWithdrawn()**" of DebitCard class returns the "**hasWithdrawn**" property using the "**return**" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **boolean** return type.

#### 4.2.5 setWithdrawalAmount()

"**setWithdrawalAmount()**" of DebitCard class is a **publicly** accessible method, used to set the "**withdrawalAmount**" property using a single parameter of type "**int**", allowing encapsulation and protected modification, it has **no return** type.

#### 4.2.6 withdraw()

"**Withdraw()**" method of DebitCard class is a **publicly** accessible method, used to withdraw a specified amount of money from a debit card, it takes in three parameters: **withdrawalAmount**, **dateofWithdrawal**, **pinNumber**, it verifies the pin number, checks the withdrawal amount against the balance amount and records the transaction, it has **no return** type.

#### 4.2.7 display()

The "**display()**" method of the DebitCard class is a **publicly** accessible method with **no return** type that **displays** account information. It checks if a withdrawal has been made, and if so, it calls the parent class's display method and prints the pin number, withdrawal

amount, and date of withdrawal. If no withdrawal has been made, it only displays the balance amount.

### 4.3 METHOD DESCRIPTIONS OF CLASS **CreditCard**

#### 4.3.1 **getCvcNumber()**

"**getCvcNumber()**" of **CreditCard** class returns the "**cvcNumber**" property using the "return" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **integer** return type.

#### 4.3.2 **getCreditLimit()**

"**getCreditLimit()**" of **CreditCard** class returns the "**creditLimit**" property using the "return" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **double** return type.

#### 4.3.3 **getInterestRate()**

"**getInterestRate()**" of **CreditCard** class returns the "**interestRate**" property using the "return" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **double** return type.

#### 4.3.4 getExpirationDate()

"**getExpirationDate()**" of CreditCard class returns the "**expirationDate**" property using the "**return**" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **String** return type.

#### 4.3.5 getGracePeriod()

"**getGracePeriod()**" of CreditCard class returns the "**gracePeriod**" property using the "return" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **integer** return type.

#### 4.3.6 getIsGranted()

"**getIsGranted()**" of CreditCard class returns the "**isGranted**" property using the "**return**" statement, allows user to access it from outside the class, it's a **publicly** accessible method with **Boolean** return type.

#### 4.3.7 setCreditLimit()

"**setCreditLimit()**" of CreditCard class, **publicly** accessible method, sets credit limit and grace period based on account balance, the method takes in two **parameters** "**creditLimit**" and "**gracePeriod**". If credit limit is less than or equal to 2.5 times the balance amount, it **sets** the credit limit and grace period and sets the value of isGranted to **true**, otherwise, it shows a message indicating that credit limit cannot be granted.

#### 4.3.8 CancelCreditCard()

"**CancelCreditCard()**" of CreditCard class, **publicly** accessible method having **no return type**, **cancels** a credit card by setting the value of cvcNumber, creditLimit, gracePeriod, and isGranted to **zero**, and **false** respectively, making the credit card unusable.

#### 4.3.9 display()

"**display()**" of CreditCard class, **publicly** accessible method, **displays** credit card information by first **calling** the parent class display method and then checking the value of isGranted, if true it **prints cvcNumber, creditLimit, gracePeriod, interestRate, and expirationDate** and if false, it only prints **cvcNumber, interestRate, and expirationDate**.

## 5. Testing(Inspection)

### 5.1 Test 1 – To inspect the DebitCard class, withdraw the amount and re-inspect the Debit Card Class

Test NO. 1	
<b>Objective:</b>	To inspect the DebitCard class, withdraw the amount and re-inspect the Debit Card Class
<b>Action:</b>	<ul style="list-style-type: none"><li>➡ The DebitCard is called with the following arguments: balanceAmount: 5000 cardId: 1234 bankAccount: "00m1234oom" issuerBank: "Everest" clientName: "Miraj" pinNumber: 5555</li><li>➡ inspection of the DebitCard class.</li><li>➡ void withdraw is called with the following arguments: withdrawalAmount: 1000 dateofWithdrawal: "2023-01-24" pinNumber: 5555</li><li>➡ Re-inspection of the DebitCard class.</li></ul>
<b>Expected Result:</b>	The balanceAmount would decrease, dateofWithdrawal, hasWithdrawn and withdrawalAmount variables would be set to the input date value, true and input withdrawn amount respectively.
<b>Actual Result:</b>	The balanceAmount was decreased, dateofWithdrawal, hasWithdrawn and withdrawalAmount variables was set to the input date value, true and input withdrawn amount respectively.
<b>Conclusion:</b>	The test is successful.

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

## OUTPUT RESULT:

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

BlueJ: Create Object

**DebitCard(int balanceAmount, int cardId, String bankAccount, String issuerBank, String clientName, int pinNumber)**

Name of Instance:

new DebitCard(  ,  
 ,  
 ,  
 ,  
 ,  
 )

OK Cancel

Figure 5: Screenshot of assigning the data in DebitCard class

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

debitCar1 : DebitCard

private int pinNumber	<input type="text" value="5555"/>	Inspect
private int withdrawalAmount	<input type="text" value="0"/>	
private String dateofWithdrawal	<input type="text" value="null"/>	Get
private boolean hasWithdrawn	<input type="text" value="false"/>	
private int cardId	<input type="text" value="1234"/>	
private String clientName	<input type="text" value="Miraj"/>	
private String issuerBank	<input type="text" value="Everest"/>	
private String bankAccount	<input type="text" value="00m1234oom"/>	
private int balanceAmount	<input type="text" value="5000"/>	

Show static fields Close

Figure 6: Screenshot for the inspection of DebitCard class

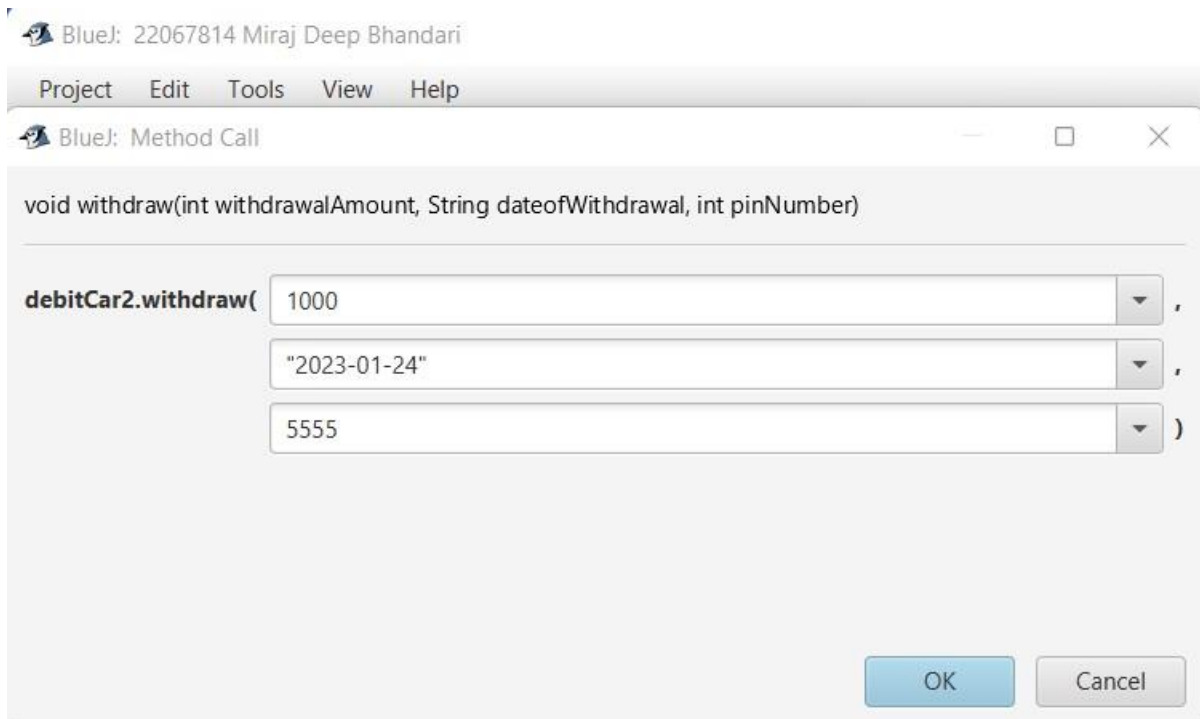


Figure 8: Screenshot of giving arguments to the withdraw method

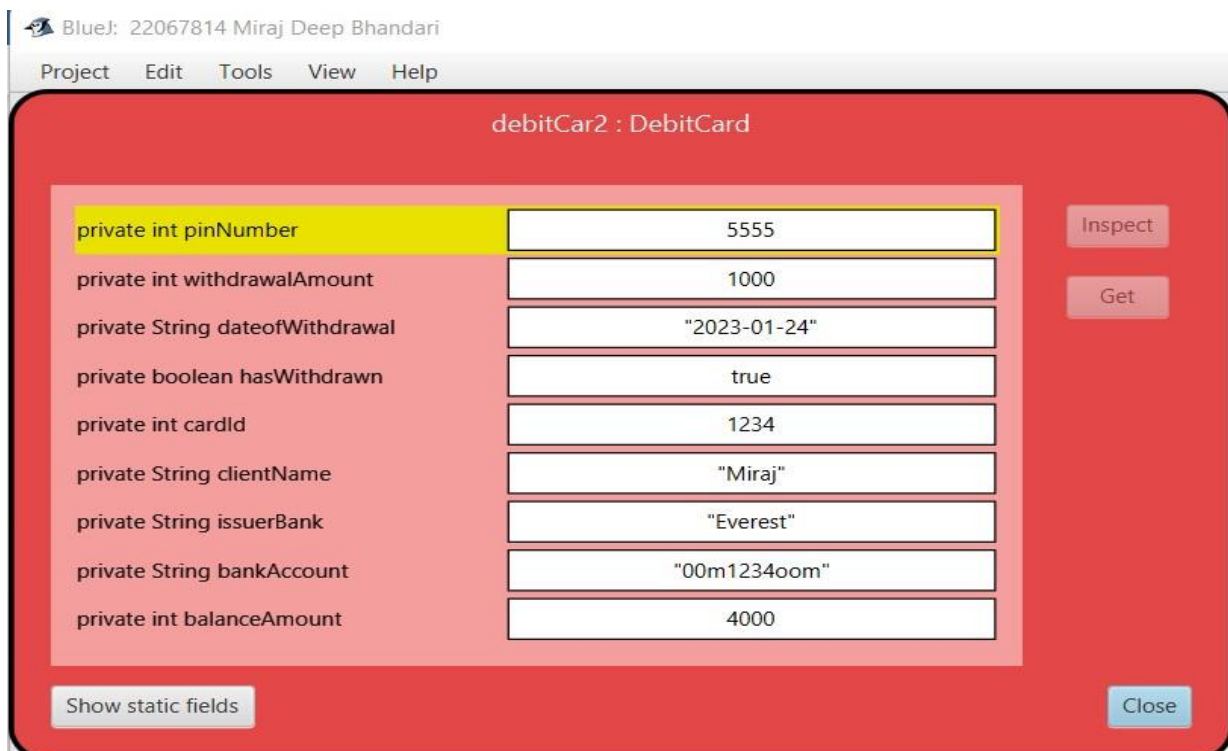


Figure 7: Screenshot of re-inspection of DebitCard class

## 5.2 Test 2 – To inspect the CreditCard class, Set the credit limit and re-inspect the CreditCard Class

Test NO.	2
<b>Objective:</b>	To inspect the CreditCard class, Set the credit limit and re-inspect the CreditCard Class
<b>Action:</b>	<p>➡ The CreditCard is called with the following arguments:</p> <p>cardId: 1234</p> <p>clientName: "Miraj"</p> <p>issuerBank: "Everest"</p> <p>bankAccount: "00m1234oom"</p> <p>balanceAmount: 5000</p> <p>cvcNumber: 2061</p> <p>interestRate: 5.2</p> <p>expirationDate: "2025-04-24"</p> <p>➡ inspection of the CreditCard class.</p> <p>➡ void setCreditLimit is called with the following arguments:</p> <p>creditLimit: 1000.0</p> <p>gracePeriod: 3</p> <p>➡ Re-inspection of the CreditCard class.</p>
<b>Expected Result:</b>	The credit limit instance variable would set to 1000.0, the gracePeriod instance variable would set to 3 and the isGranted variable would set to true.
<b>Actual Result:</b>	The credit limit instance variable was set to 1000.0, the gracePeriod instance variable was set to 3 and the isGranted variable was set to true.
<b>Conclusion:</b>	The test is successful.

Table 2: To inspect the CreditCard class, Set the credit limit and re-inspect the CreditCard Class



## OUTPUT RESULT:

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

BlueJ: Create Object

**CreditCard(int cardId, String clientName, String issuerBank, String bankAccount, int balanceAmount, int cvcNumber, double interestRate, String expirationDate)**

Name of Instance:

new CreditCard(

<input type="text" value="1234"/>	,
<input type="text" value="Miraj"/>	,
<input type="text" value="Everest"/>	,
<input type="text" value="00m1234oom"/>	,
<input type="text" value="5000"/>	,
<input type="text" value="2061"/>	,
<input type="text" value="5.2"/>	,
<input type="text" value="2025-04-24"/>	)

OK Cancel

Figure 9: Screenshot of assigning the data in CreditCard class

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

creditCa2 : CreditCard

private int cvcNumber	<input type="text" value="2061"/>	Inspect
private double creditLimit	<input type="text" value="0.0"/>	
private double interestRate	<input type="text" value="5.2"/>	Get
private String expirationDate	<input type="text" value="2025-04-24"/>	
private int gracePeriod	<input type="text" value="0"/>	
private boolean isGranted	<input type="text" value="false"/>	
private int cardId	<input type="text" value="1234"/>	
private String clientName	<input type="text" value="Miraj"/>	
private String issuerBank	<input type="text" value="Everest"/>	
private String bankAccount	<input type="text" value="00m1234oom"/>	
private int balanceAmount	<input type="text" value="5000"/>	

Show static fields Close

Figure 10: Screenshot for the inspection of CreditCard class

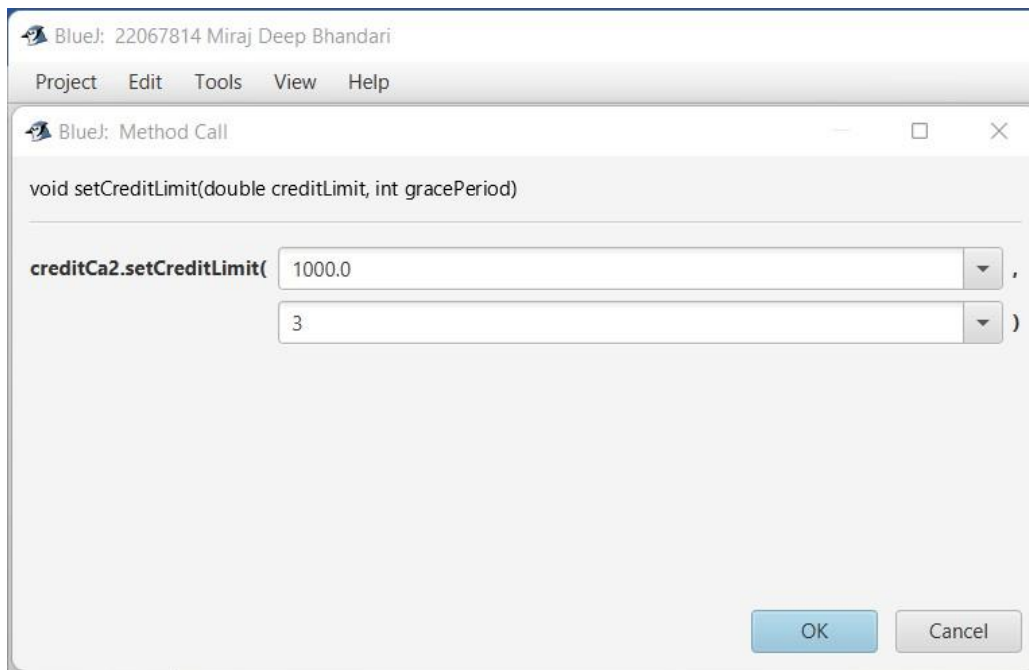


Figure 11: Screenshot of giving arguments to the setCreditLimit method

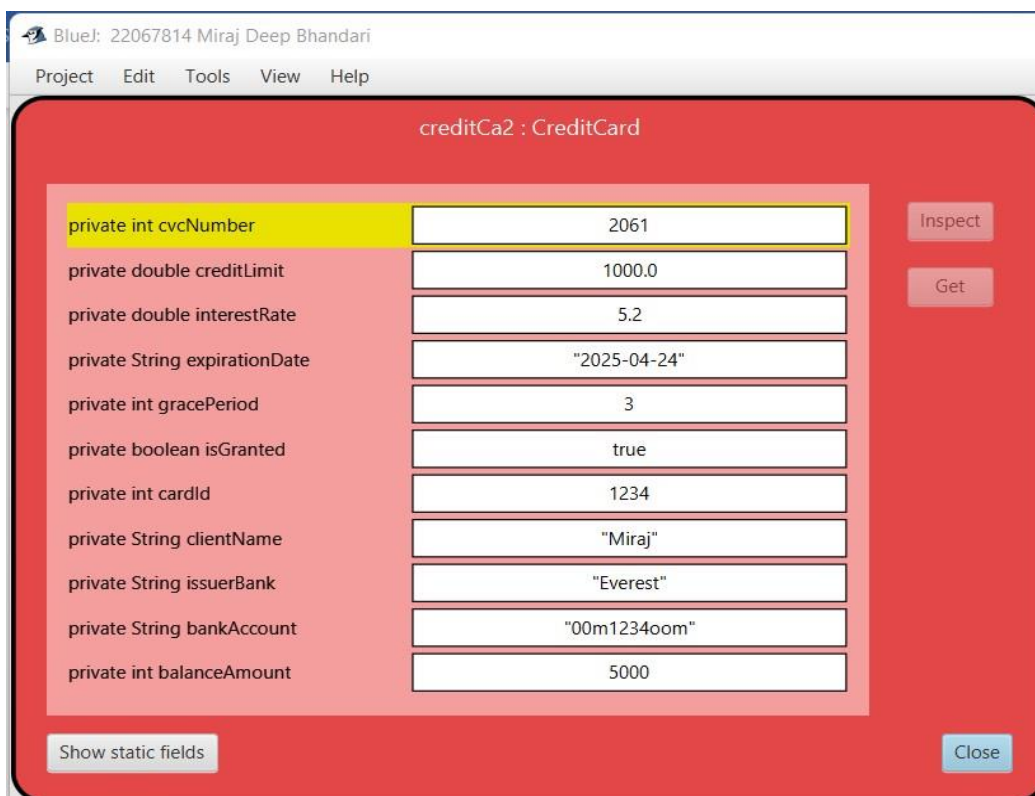


Figure 12: Screenshot of re-inspection of CreditCard class

### 5.3 Test 3 – To inspect the CreditCard class again after cancelling the credit card.

Test NO. 3	
<b>Objective:</b>	To inspect the CreditCard class again after cancelling the credit card.
<b>Action:</b>	<p>➡ The CreditCard is called with the following arguments:  cardId: 3456  clientName: "Arbit"  issuerBank: "Everest"  bankAccount: "00m1234oom"  balanceAmount: 6000  cvcNumber: 2060  interestRate: 15.2  expirationDate: "2025-04-24"</p> <p>➡ void setCreditLimit is called with the following arguments:  creditLimit: 1200.0  gracePeriod: 4</p> <p>➡ inspection of the CreditCard class before cancelling credit card.</p> <p>➡ void cancelCreditCard is called.</p> <p>➡ Re-inspection of the CreditCard class.</p>
<b>Expected Result:</b>	The value of cvcNumber would be 0, creditLimit would be 0.0, gracePeriod would be 0, and isGranted would be false.
<b>Actual Result:</b>	The value of cvcNumber was 0, creditLimit was 0.0, gracePeriod was 0, and isGranted was false.
<b>Conclusion:</b>	The test is successful.

*Table 3: To inspect the CreditCard class again after cancelling the credit card.*

## OUTPUT RESULT:

The screenshot shows the 'Create Object' dialog for the `CreditCard` class. The dialog has a title bar 'BlueJ: 22067814 Miraj Deep Bhandari' and a menu bar with 'Project', 'Edit', 'Tools', 'View', and 'Help'. The main area displays the class signature `CreditCard(int cardId, String clientName, String issuerBank, String bankAccount, int balanceAmount, int cvcNumber, double interestRate, String expirationDate)`. Below this, the 'Name of Instance:' field contains 'creditCa1'. The 'new CreditCard(' prefix is followed by eight input fields, each with a dropdown arrow. The values entered are: '3456', '"Arbit"', '"Everest"', '"00m1234oom"', '6000', '2060', '15.2', and '"2025-04-24"'. The dialog ends with a closing parenthesis ')'. At the bottom right are 'OK' and 'Cancel' buttons.

*Figure 13: Screenshot of assigning the data in CreditCard class*

The screenshot shows the 'Method Call' dialog for the `setCreditLimit` method. The dialog has a title bar 'BlueJ: 22067814 Miraj Deep Bhandari' and a menu bar with 'Project', 'Edit', 'Tools', 'View', and 'Help'. The main area displays the method signature `void setCreditLimit(double creditLimit, int gracePeriod)`. Below this, the 'creditCa1.setCreditLimit(' prefix is followed by two input fields, each with a dropdown arrow. The values entered are '1200.0' and '4'. The dialog ends with a closing parenthesis ')'. At the bottom right are 'OK' and 'Cancel' buttons.

*Figure 14: Screenshot of giving arguments to the setCreditLimit method*

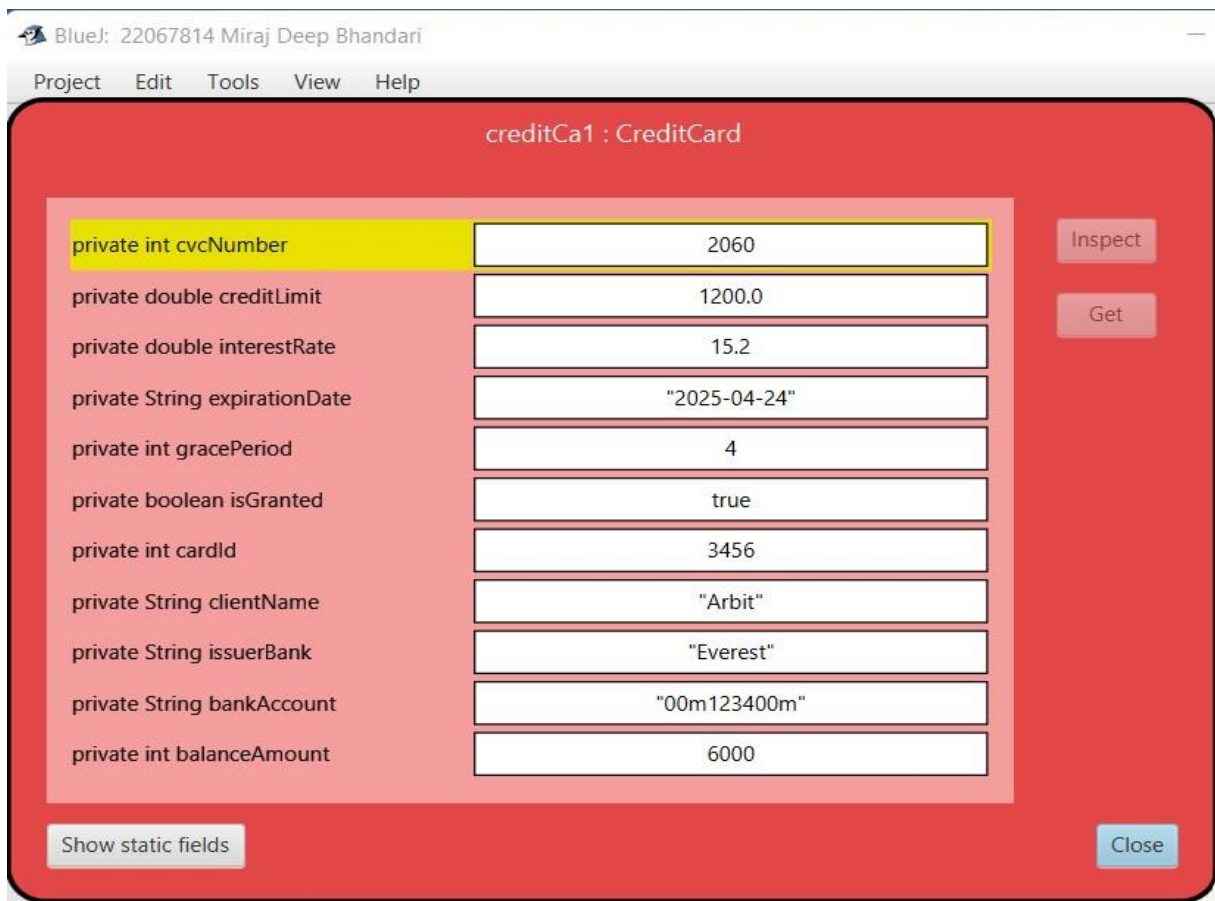


Figure 15: Screenshot of inspection of creditcard before cancelling credit card

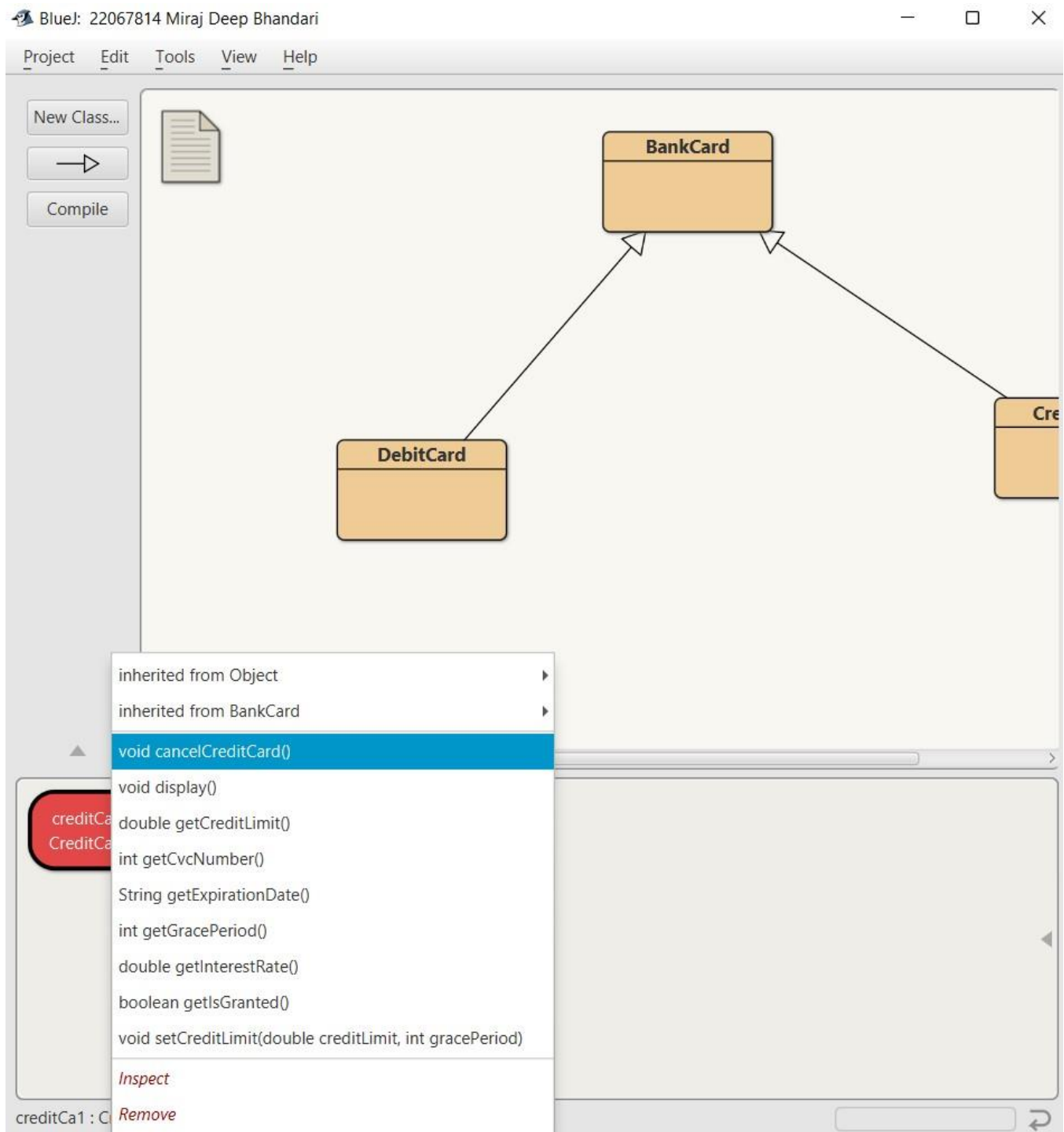


Figure 16: Screenshot of cancelling Credit Card

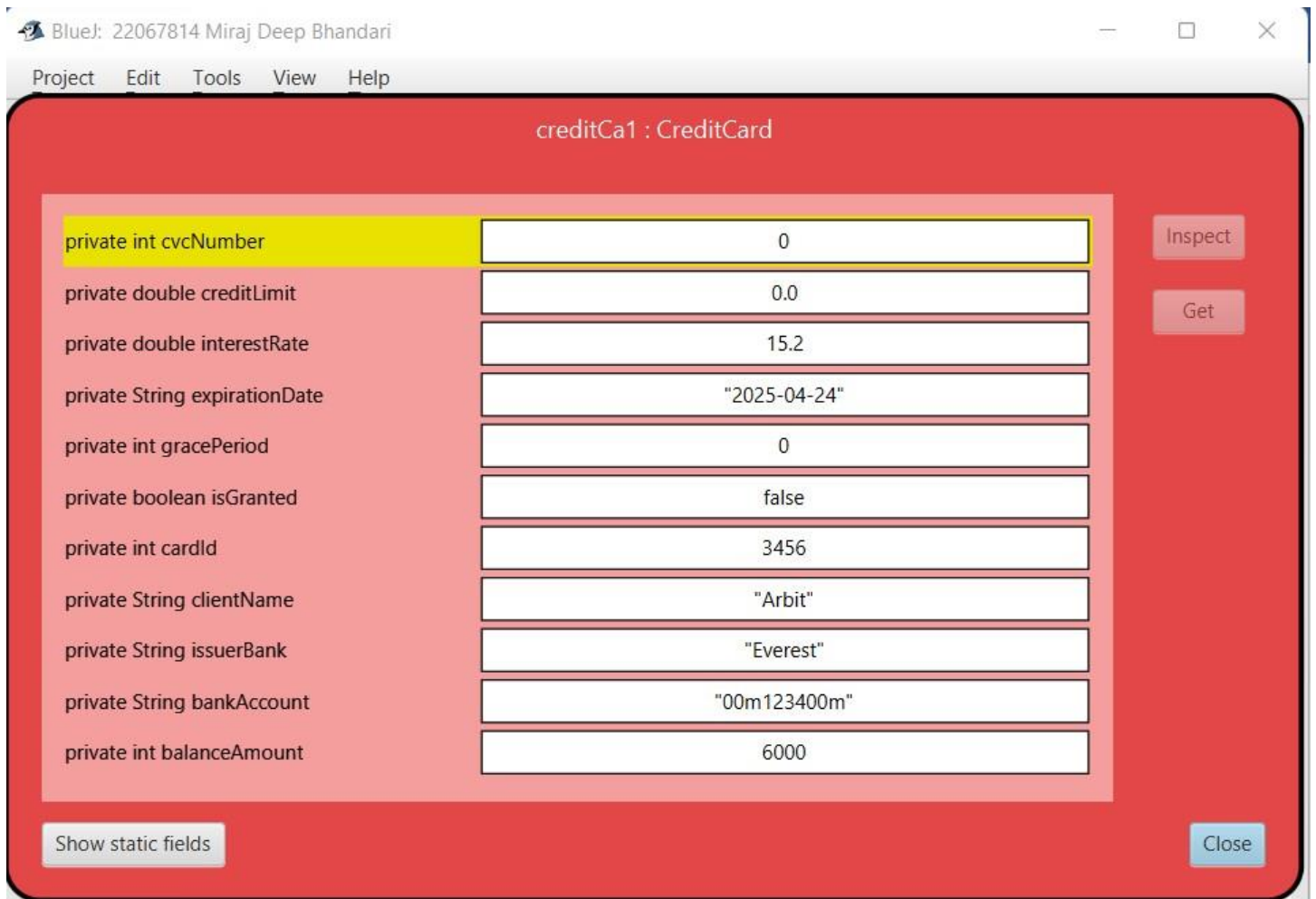


Figure 17: Screenshot of re-inspection of CreditCard class after cancelling credit class

#### 5.4 Test 4 – To display the details of DebitCard and CreditCard classes.

Test NO.	4
Objective:	To display the details of DebitCard and CreditCard classes.
Action:	<p>➡ The DebitCard is called with the following arguments: balanceAmount: 5000 cardId: 1234 bankAccount: "00m1234oom" issuerBank: "Everest" clientName: "Miraj" pinNumber: 5555</p> <p>➡ void display is called in DebitCard class before withdraw</p> <p>➡ void withdraw is called in DebitCard class with the following arguments: withdrawalAmount: 1000 dateofWithdrawal: "2023-1-22" pinNumber: 5555</p> <p>➡ void display is called in DebitCard class after withdraw</p> <p>➡ The CreditCard is called with the following arguments: cardId: 1234 clientName: "Miraj" issuerBank: "Everest" bankAccount: "00m1234oom" balanceAmount: 5000 cvcNumber: 2061 interestRate: 5.2 expirationDate: "2025-04-24"</p> <p>➡ void display is called in CreditCard class before setting Credit</p>



	<p>Limit</p> <p>➡ void setCreditLimit is called with the following arguments: creditLimit: 1000.0 gracePeriod: 3</p> <p>➡ void display is called in CreditCard class after setting Credit Limit</p>
<b>Expected Result:</b>	<p><u>For DebitCard Class</u></p> <p>The value of balanceAmount would display as output when I call display method before withdraw.</p> <p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, pinNumber, withdrawalAmount and dateofWithdrawal would display as output respectively when display method is called after withdraw.</p> <p><u>For CreditCard Class</u></p> <p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, cvcNumber, interestRate and expirationDate would display as output respectively when I call display method before setting credit limit.</p> <p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, cvcNumber, creditLimit, gracePeriod, interestRate and expirationDate would display as output respectively when I call display method after setting credit limit.</p>
<b>Actual Result:</b>	<p><u>For DebitCard Class</u></p> <p>The value of balanceAmount was displayed as output when I call display method before withdraw.</p>

	<p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, pinNumber, withdrawalAmount and dateofWithdrawal was displayed as output respectively when I call display method after withdraw.</p> <p><u>For CreditCard Class</u></p> <p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, cvcNumber, interestRate and expirationDate was displayed as output respectively when I call display method before setting credit limit.</p> <p>The value of Clientname, CardId, issuerBank, bankAccount, balanceAmount, cvcNumber, creditLimit, gracePeriod, interestRate and expirationDate was displayed as output respectively when I call display method after setting credit limit</p>
<b>Conclusion:</b>	The test is successful.

*Table 4: To display the details of DebitCard and CreditCard classes.*

## OUTPUT RESULT:

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

BlueJ: Create Object

**DebitCard(int balanceAmount, int cardId, String bankAccount, String issuerBank, String clientName, int pinNumber)**

Name of Instance:

**new DebitCard(**  ,  
 ,  
 ,  
 ,  
 ,  
 )

OK Cancel

Figure 18: Screenshot of assigning the data in DebitCard class

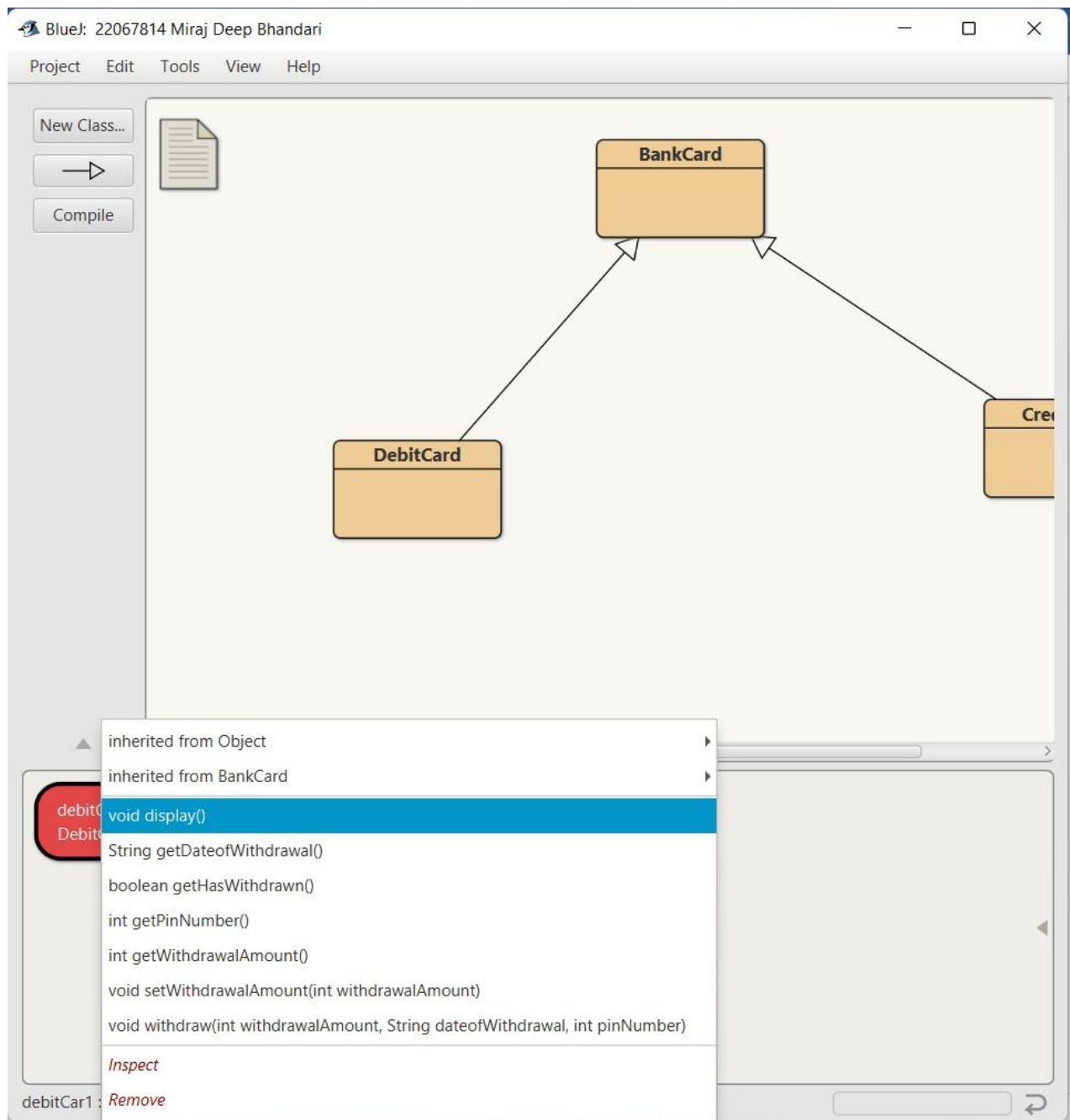
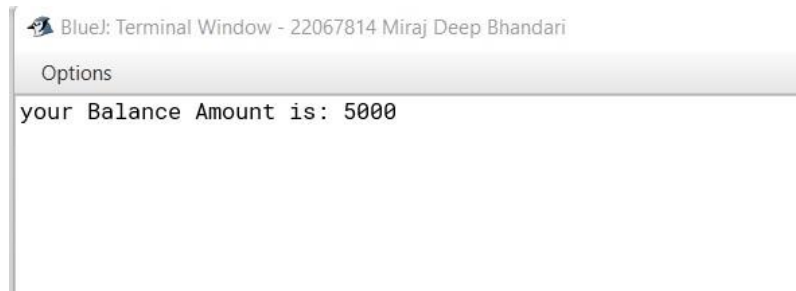
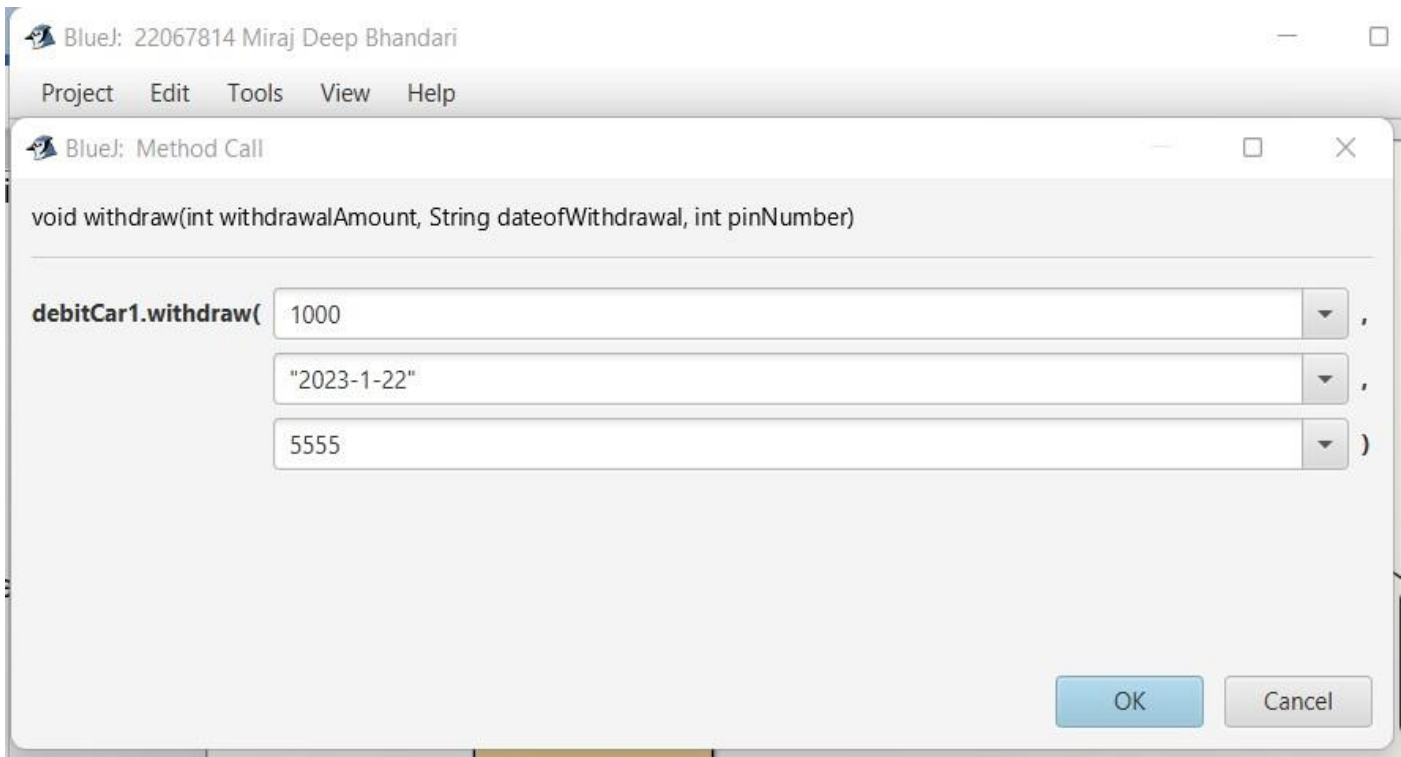


Figure 19: Screenshot of calling display method before withdraw



*Figure 20: Screenshot of output of display method before withdraw*



*Figure 21: Screenshot of giving values to the withdraw method*

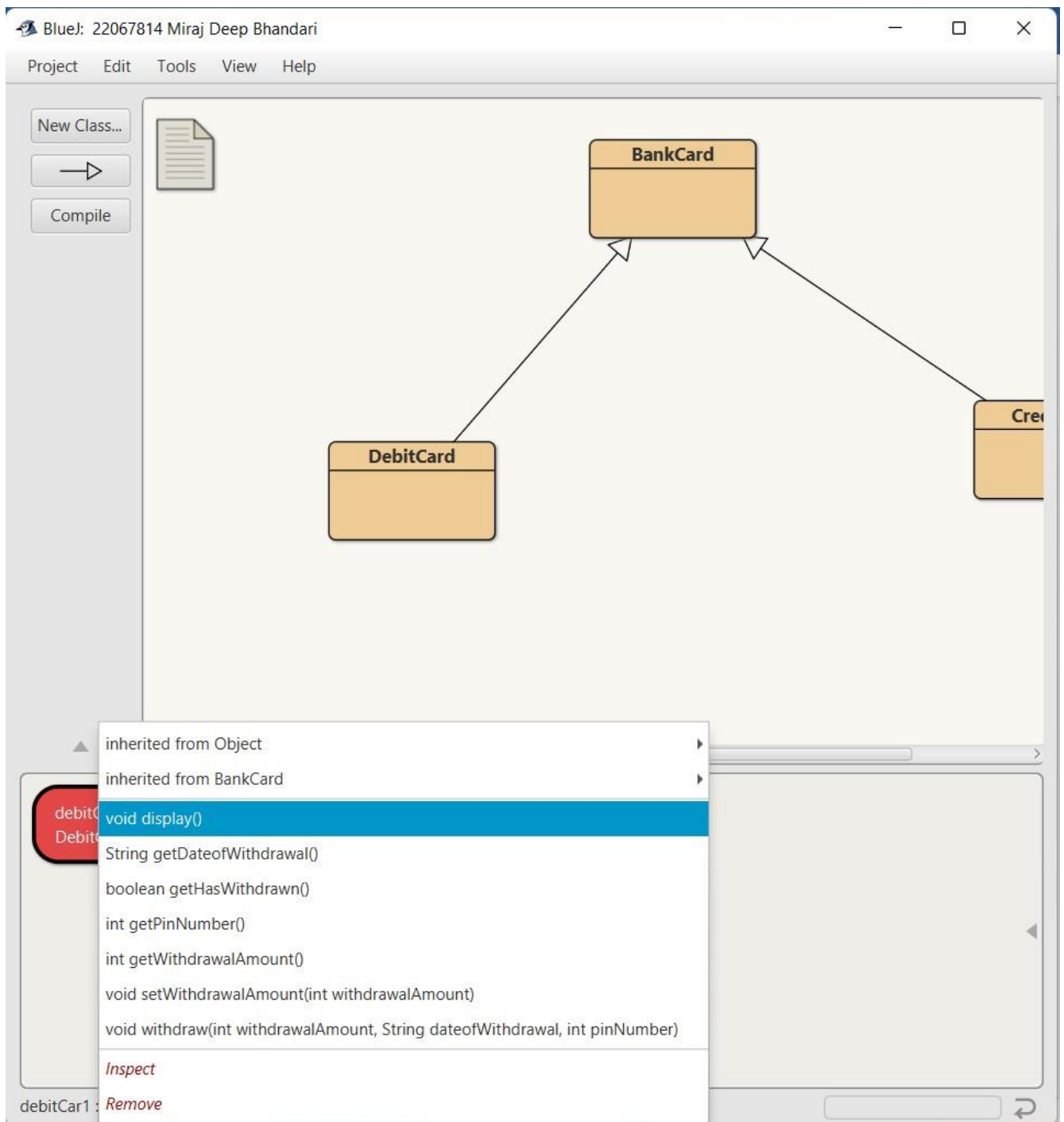
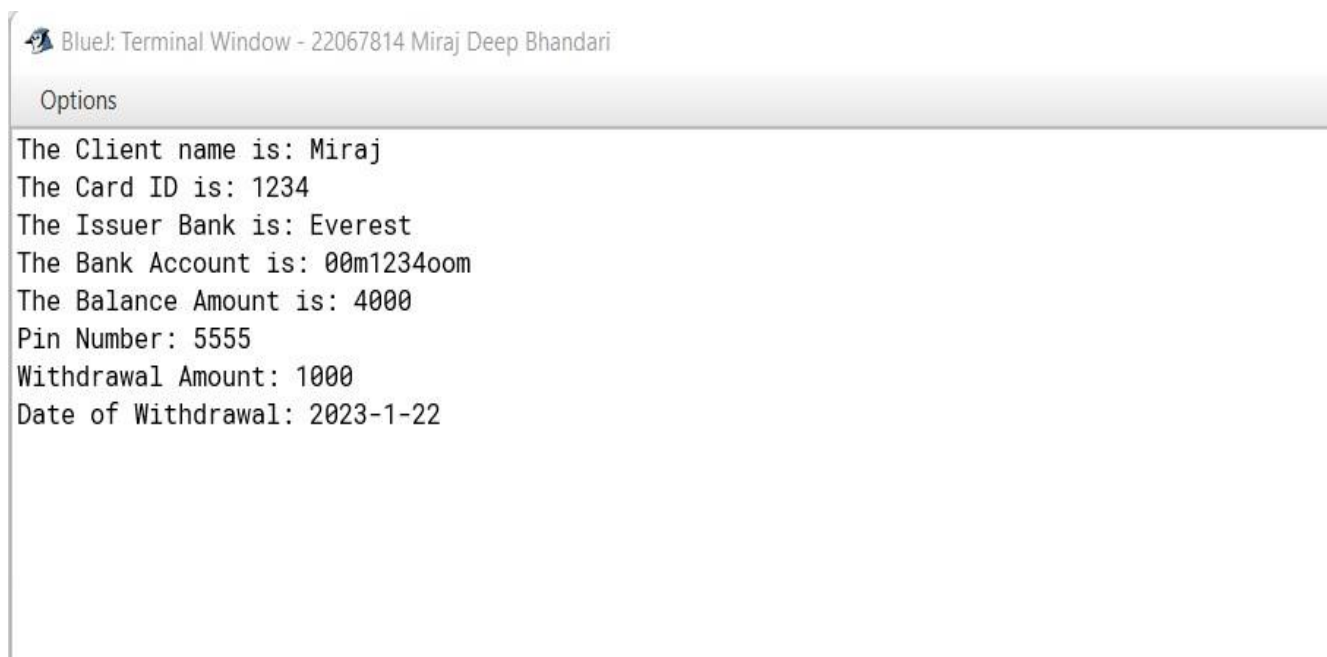


Figure 22: Screenshot of calling display method after withdraw



*Figure 23: Screenshot of output of display method after withdraw*

BlueJ: 22067814 Miraj Deep Bhandari

Project Edit Tools View Help

BlueJ: Create Object

**CreditCard(int cardId, String clientName, String issuerBank, String bankAccount, int balanceAmount, int cvcNumber, double interestRate, String expirationDate)**

Name of Instance:

**new CreditCard(**  **,**  **,**  **,**  **,**  **,**  **,**  **,**  **)**

OK Cancel

Figure 24: Screenshot of assigning the data in CreditCard class



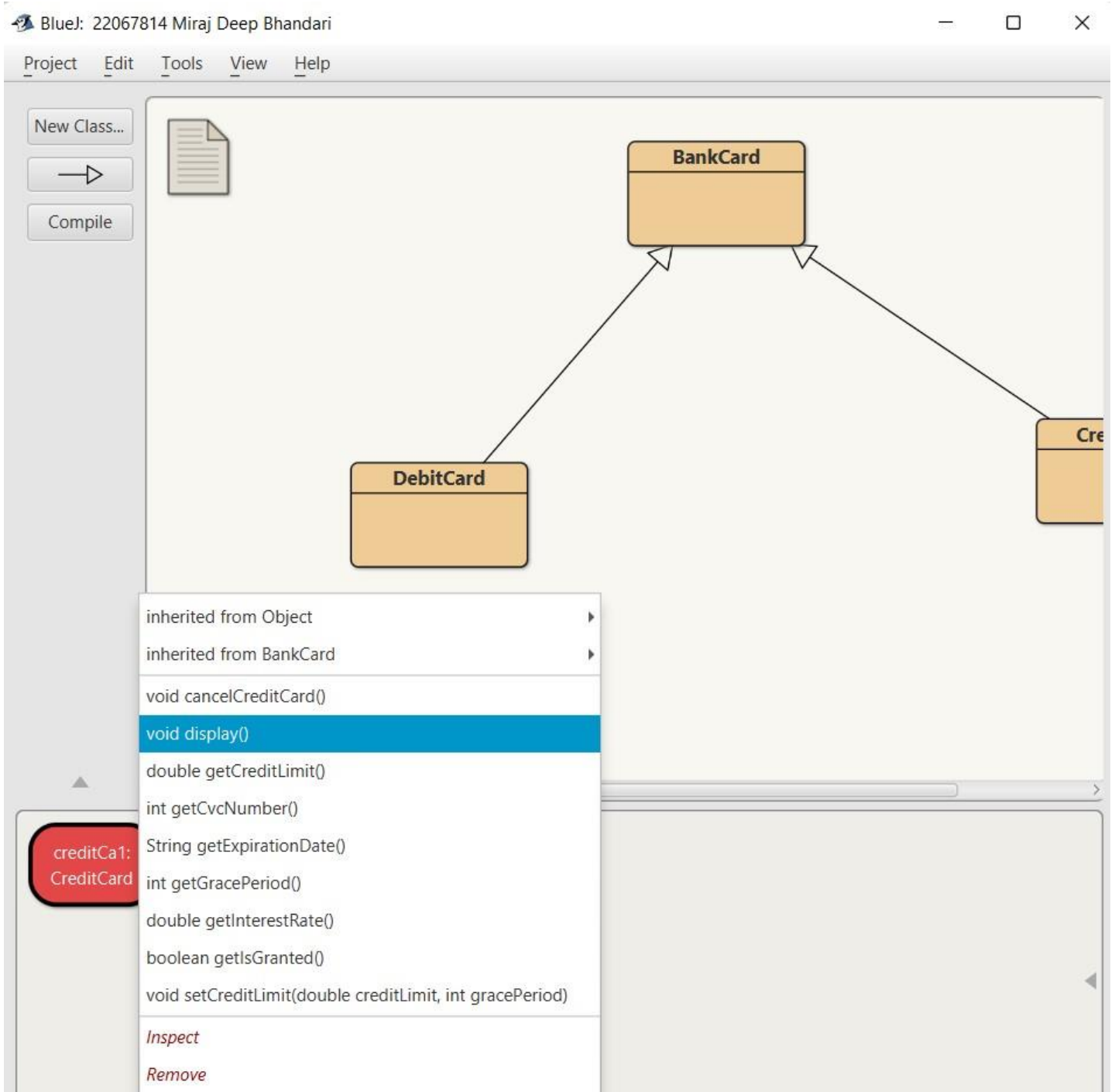


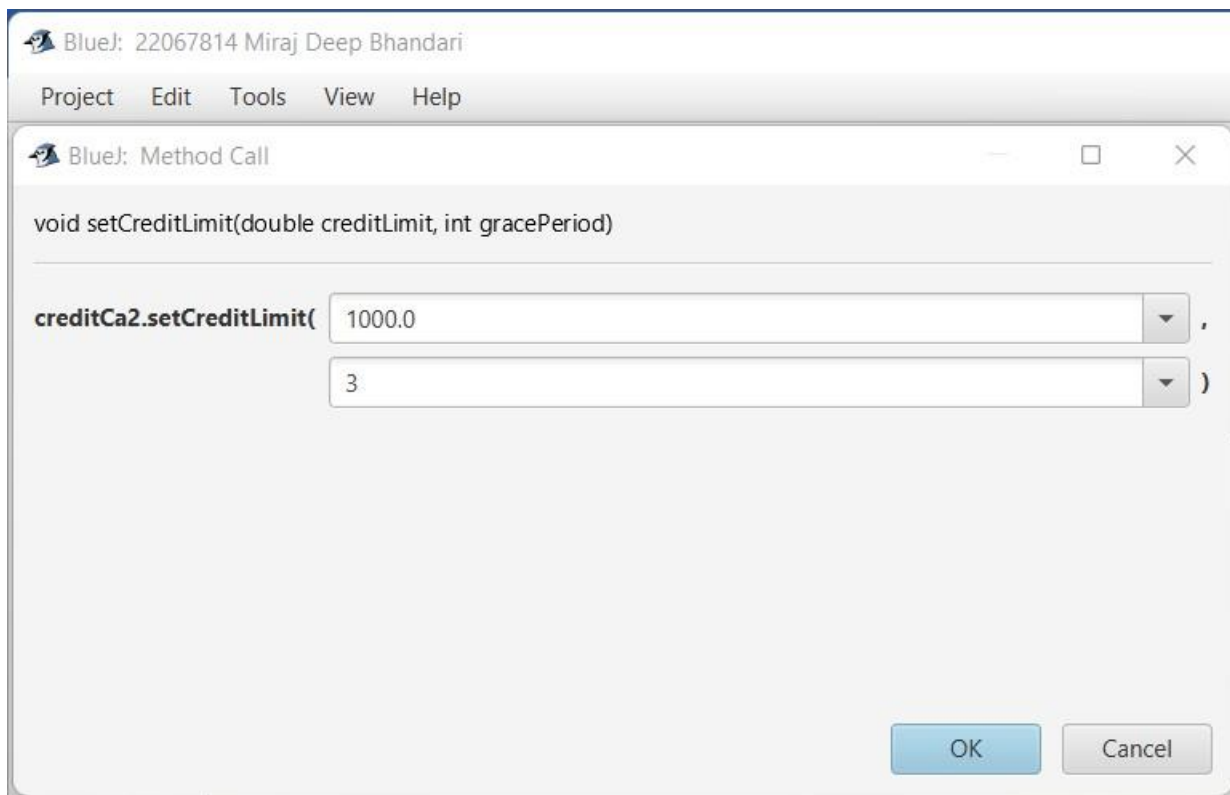
Figure 25: Screenshot of calling display method before setting credit limit

BlueJ: Terminal Window - 22067814 Miraj Deep Bhandari

#### Options

```
The Client name is: Miraj
The Card ID is: 1234
The Issuer Bank is: Everest
The Bank Account is: 00m1234oom
The Balance Amount is: 5000
Your CVC Number: 2061
Your Interest Rate: 5.2
Your Expiration Date: 2025-04-24
```

*Figure 26: Screenshot of output of display method before setting credit limit*



*Figure 27: Screenshot of giving values to setCreditLimit*

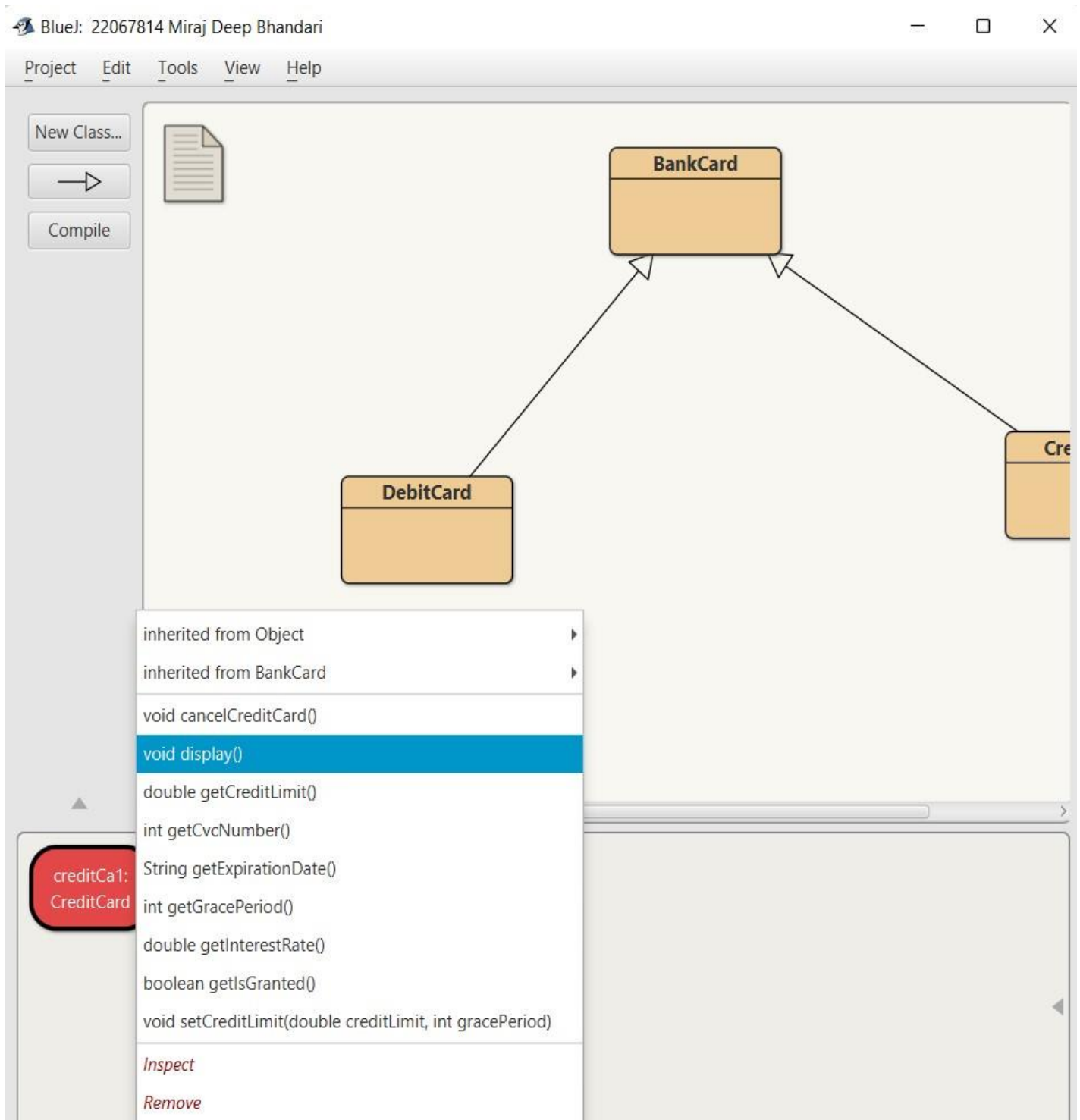


Figure 28: Screenshot of calling display method after setting credit limit

BlueJ: Terminal Window - 22067814 Miraj Deep Bhandari

#### Options

```
The Client name is: Miraj
The Card ID is: 1234
The Issuer Bank is: Everest
The Bank Account is: 00m1234oom
The Balance Amount is: 5000
Your CVC Number: 2061
Your Credit Limit: 1000.0
Your Grace Period: 3
Your Interest Rate: 5.2
Your Expiration Date: 2025-04-24
```

*Figure 29: Screenshot of output of display method after setting credit limit*

## 6. ERROR DETECTION AND CORRECTION

A Java program can malfunction due to an illegal statement or operation, which is referred to as an error (Mishra, 2022). There are main three types of common errors in java programming which are listed below:

### 1) Syntax Errors:

In Java programming, syntax errors occur when the code does not conform to the proper structure or format of the Java language. These errors can include issues such as missing semicolons, incorrect use of parentheses, or using a reserved word as a variable name. The compiler will not be able to understand and execute the code, and will generate an error message indicating the location and nature of the syntax error.

### 2) Logical Errors:

In Java programming, logical errors are mistakes in the code logic that do not prevent the program from compiling, but cause it to produce incorrect or unexpected results.

Logical errors can occur due to a variety of issues, such as:

- Using the wrong operator or operator precedence
- Not initializing a variable before using it
- Not updating a variable value when it needs to change
- Not using the correct loop or conditional construct
- Misunderstanding the problem or requirements
- Not testing the code thoroughly

Logical errors can be difficult to detect and fix because the program will not generate any error messages, but will produce incorrect output. They require careful debugging and testing to identify and fix (Mishra, 2022).

### **3)Runtime Errors:**

In Java programming, a runtime error is an error that occurs during the execution of a program, rather than during the compilation process. These errors are caused by issues such as attempting to access an array index that is out of bounds, dividing a number by zero, attempting to access a null object, running out of memory or by trying to open a file that doesn't exist (Mishra, 2022).

Errors are an inherent aspect of software development, particularly when working with complex programming languages such as Java. As a developer, I have had the opportunity to work with Java in the BlueJ development environment, where I have encountered a variety of errors, including both syntax and logical errors. Initially, identifying and resolving these errors proved to be quite challenging. However, through a process of research and consultation with peers, the technical community and my instructor, I was able to gain a deeper understanding of the underlying causes of these errors and develop effective strategies for debugging and correcting my code. It was a learning experience for me, which helped me to become a more proficient developer. It was not an easy task but the satisfaction of solving the errors and making the code work was worth it. The specific errors encountered, as well as the solutions implemented, are detailed below:

## 6.1 FIRST ERROR AND ITS SOLUTION.

### ERROR TYPE: LOGICAL ERROR

```
public class BankCard {  
  
    private int cardId;  
    private String clientName;  
    private String issuerBank;  
    private String bankAccount;  
    private int balanceAmount;  
  
    public BankCard(int balanceAmount, int cardId, String bankAccount, String issuerBank)  
    {  
        balanceAmount = balanceAmount;  
        cardId = cardId;  
        bankAccount = bankAccount;  
        issuerBank = issuerBank;  
        clientName = "";  
    }  
}
```

*Figure 30: First Error (logical error)*

The error which is shown above is a **logical error**. the error can be found inside the constructor. In the constructor I **forgot to use this keyword** so doing `balanceAmount = balanceAmount`, `cardId = cardId`, `issuerBank = IssuerBank` and `clientName = ""` parameters variables was not being assigned to the instance variables and Instance variable client name was not set to empty . **local variable** is **different** from the **instance variable**, so the program would not be able to **refer to the instance variable**, and it would not work as intended. This would cause the program to **execute incorrectly**, and it would produce **unexpected results**



## SOLUTION OF THE ERROR:

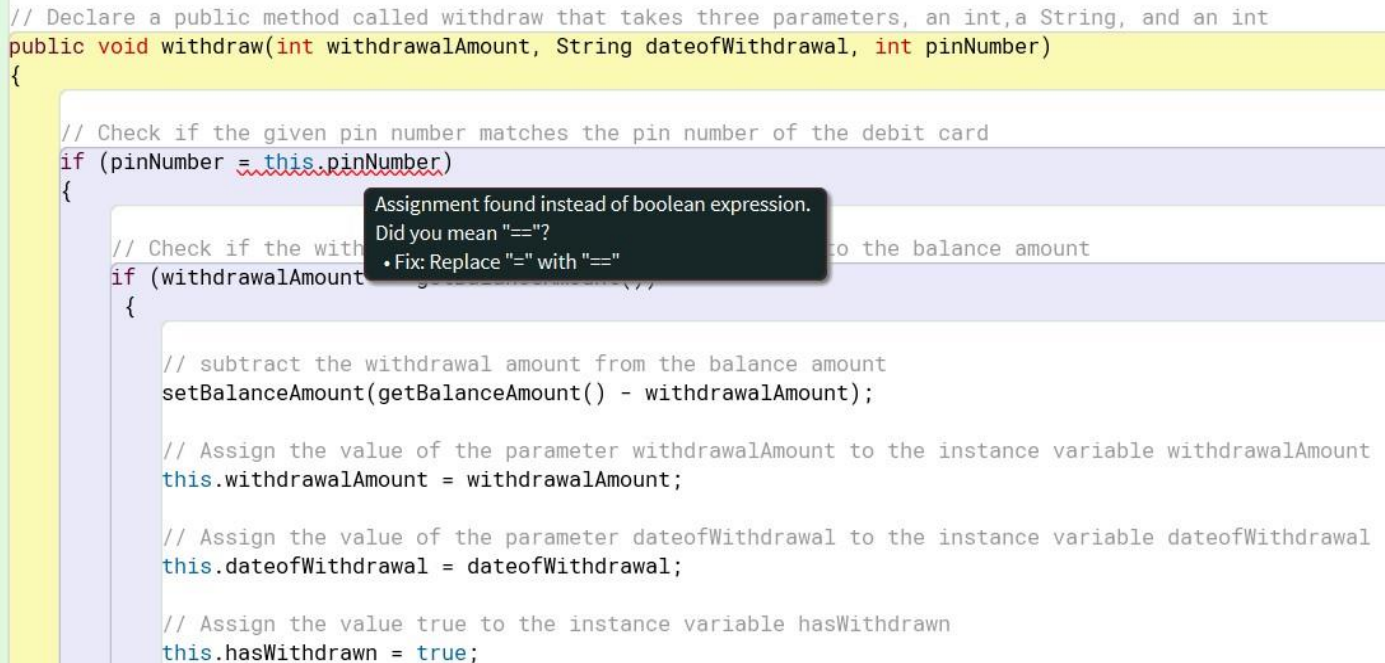
```
public class BankCard {  
  
    private int cardId;  
    private String clientName;  
    private String issuerBank;  
    private String bankAccount;  
    private int balanceAmount;  
  
    public BankCard(int balanceAmount, int cardId, String bankAccount, String issuerBank)  
    {  
        this.balanceAmount = balanceAmount;  
        this.cardId = cardId;  
        this.bankAccount = bankAccount;  
        this.issuerBank = issuerBank;  
        this.clientName = "";  
    }  
}
```

*Figure 31:Solution of the first error (logical error)*

I was able to resolve this issue by utilizing the **"this"** keyword. The "this" keyword helped the compiler understand that I was **referring** to the **instance variable**, and as a result, the value of the **parameter variable** was **assigned** to the **instance variable**, and the value of the instance variable **"clientName"** was set to an **empty** value.

## 6.2 SECOND ERROR AND ITS SOLUTION.

### ERROR TYPE: SYNTAX ERROR



```
// Declare a public method called withdraw that takes three parameters, an int, a String, and an int
public void withdraw(int withdrawalAmount, String dateofWithdrawal, int pinNumber)
{
    // Check if the given pin number matches the pin number of the debit card
    if (pinNumber = this.pinNumber)
    {
        // Check if the withdrawal amount is greater than the balance amount
        if (withdrawalAmount > getBalanceAmount())
        {
            // subtract the withdrawal amount from the balance amount
            setBalanceAmount(getBalanceAmount() - withdrawalAmount);

            // Assign the value of the parameter withdrawalAmount to the instance variable withdrawalAmount
            this.withdrawalAmount = withdrawalAmount;

            // Assign the value of the parameter dateofWithdrawal to the instance variable dateofWithdrawal
            this.dateofWithdrawal = dateofWithdrawal;

            // Assign the value true to the instance variable hasWithdrawn
            this.hasWithdrawn = true;
        }
    }
}
```

*Figure 32: Second Error (Syntax error)*

The error displayed above was identified as a Syntax error. Upon further analysis, it was determined that the error occurred within an If statement, specifically in the condition provided. Upon further examination, it was determined that the error was a result of a typographical mistake, where the assignment operator “=” was used in place of the Equal to operator “==”.

## SOLUTION OF THE ERROR

```
// Declare a public method called withdraw that takes three parameters, an int, a String, and an int
public void withdraw(int withdrawalAmount, String dateofWithdrawal, int pinNumber)
{
    // Check if the given pin number matches the pin number of the debit card
    if (pinNumber == this.pinNumber)
    {
        // Check if the withdrawal amount is less than or equal to the balance amount
        if (withdrawalAmount <= getBalanceAmount())
        {
            // subtract the withdrawal amount from the balance amount
            setBalanceAmount(getBalanceAmount() - withdrawalAmount);

            // Assign the value of the parameter withdrawalAmount to the instance variable withdrawalAmount
            this.withdrawalAmount = withdrawalAmount;

            // Assign the value of the parameter dateofWithdrawal to the instance variable dateofWithdrawal
            this.dateofWithdrawal = dateofWithdrawal;

            // Assign the value true to the instance variable hasWithdrawn
            this.hasWithdrawn = true;
        }
    }
}
```

*Figure 33: Solution of the second error (Syntax error)*

Upon further inspection of my code, I discovered that I had made an error in my **assignment operator usage**. Specifically, I had utilized the "=" operator instead of the "==" operator. This led to an error in my code, as the "=" operator is used for assignment and the "==" operator is used for **comparison**.

To **rectify** this issue, I replaced the "=" operator with the "==" operator in the appropriate location. This modification eliminated the error, as indicated by the **disappearance** of the **error underline**.

### 6.3 THIRD ERROR AND ITS SOLUTION.

#### ERROR TYPE:

```
//constructor of CreditCard class
public CreditCard(int cardId, String clientName, String issuerBank, String bankAccount,
int balanceAmount, int cvcNumber, double interestRate, String expirationDate)
{

    // Call the constructor of the parent class and pass the parameters
    super(int balanceAmount, int cardId, String bankAccount, String issuerBank);

    // Call the setter method to set the clientName
    setClientName(clientName);

    // Assign the parameter cvcNumber to the instance variable cvcNumber
    this.cvcNumber = cvcNumber;

    // Assign the parameter interestRate to the instance variable interestRate
    this.interestRate = interestRate;
}
```

*Figure 34: Third Error (Syntax error)*

I found another **syntax error** while writing java code in the bluej. The error shown in above figure is due to the keyword "**super**" is followed by a set of parentheses and **variables**, like in "super(int balanceAmount, int cardId, String bankAccount, String issuerBank);" This is not the correct syntax for calling a constructor from a parent class.

## SOLUTION OF THE ERROR

```
//constructor of CreditCard class
public CreditCard(int cardId, String clientName, String issuerBank, String bankAccount,
int balanceAmount,int cvcNumber, double interestRate, String expirationDate)
{

    // Call the constructor of the parent class and pass the parameters
    super( balanceAmount, cardId, bankAccount, issuerBank);

    // Call the setter method to set the clientName
    setClientName(clientName);

    // Assign the parameter cvcNumber to the instance variable cvcNumber
    this.cvcNumber = cvcNumber;

    // Assign the parameter interestRate to the instance variable interestRate
    this.interestRate = interestRate;
}
```

*Figure 35: Solution of the Third Error (Syntax error)*

After identifying the syntax error displayed above, I took the necessary steps to rectify it by **removing** the **data types int** and **string** which were mistakenly included **inside** the **parentheses** of the "super" keyword. I discovered that the correct syntax for calling a constructor from a parent class is to use the keyword "super" followed by a set of parentheses, with **no variables inside**. While it is appropriate to pass parameters to the parent class constructor, it is not correct to include variables inside the parentheses of the "**super**" keyword.

## 6.4 FOUR ERROR AND ITS SOLUTION.

### ERROR TYPE: SEMANTIC ERROR

```
public class BankCard {  
  
    private int cardId;  
    private String clientName;  
    private String issuerBank;  
    private String bankAccount;  
    private int balanceAmount;  
  
    // creating constructor with four arguments: balanceAmount, cardId, bankAccount, and issuerBank.  
    public BankCard(int balanceAmount, int cardId, String bankAccount, String issuerBank)  
    {  
  
        this.balanceAmount = balanceAmount;  
        this.CardId = cardId;  
        this.bankAc
```

cannot find symbol - variable CardId

*Figure 36: Fourth Error (Semantic error)*

I found **semantic error** while writing java code in the bluej. The semantic error that I encountered in your Java code in BlueJ is due to the use of an **undeclared** variable "**CardId**". In the constructor, I was trying to set the value of an instance variable cardId with CardId which is not declared in the class.

## SOLUTION OF THE ERROR

```
public class BankCard {  
  
    private int cardId;  
    private String clientName;  
    private String issuerBank;  
    private String bankAccount;  
    private int balanceAmount;  
  
    // creating constructor with four arguments: balanceAmount, cardId, bankAccount, and issuerBank.  
    public BankCard(int balanceAmount, int cardId, String bankAccount, String issuerBank)  
    {  
  
        this.balanceAmount = balanceAmount;  
        this.cardId = cardId;  
        this.bankAccount = bankAccount;  
    }  
}
```

*Figure 37: Solution of the fourth error (Semantic error)*

I fixed the **semantic** mistake in my code, by changing "**CardId**" to "cardId" in the constructor of the class. This was necessary as "cardId" is the correct spelling for the instance variable that I had previously declared. This resolved the error that occurred because I had mistakenly used an undeclared variable "**CardId**".

## 7. CONCLUSION

The completion of this project has been a significant accomplishment for me, as it highlights my ability to adapt to new programming languages and technologies. My prior experience had been primarily in **C**, a **procedural programming** language, which required a different approach to problem-solving and code organization than Java, an object-oriented programming language. The project has given me a strong foundation in object-oriented programming concepts and techniques, particularly through the utilization of **classes**, **objects**, **inheritance**, and **encapsulation**.

The **implementation** of **classes** and **objects** in Java has enabled me to **model real-world** entities and their behavior in a more efficient and organized manner. Moreover, the incorporation of **inheritance** and **encapsulation** has permitted me to establish a **hierarchical structure** within my code, making it more **modular** and **reusable**. This has resulted in a more robust and maintainable codebase.

This project has also provided me with valuable experience in working with my **instructors and teachers**, who have been instrumental in guiding me throughout the process. **Report making** and **communication** are essential skills in this project and this project has allowed me to practice and improve these skills. I am excited to continue to develop my skills in Java programming and apply what I have learned to future projects.

As a novice in the realm of object-oriented programming, I encountered a plethora of obstacles while working on my project. These difficulties primarily manifested in the form of logical and syntax errors. Syntax errors, although present, were relatively simple to



rectify through proper analysis of the code and program flow. I was able to identify and fix these errors by studying the program and referring to various online resources. I also familiarized myself with the common syntax errors and how to fix them, which helped me to avoid similar mistakes in the future. Despite my efforts, some logical errors proved to be more challenging and required me to seek guidance from my instructors and teachers.

Logical errors, unlike syntax errors, required a more in-depth understanding of the program and its flow. I attempted to solve these errors by analyzing the code and the flow of the program, but there were some errors that were new to me and I had to seek help from my teachers and instructors. These errors were more complex and required a deeper understanding of the concepts and principles of object-oriented programming. I had to consult various resources and seek the guidance of experts to fully understand the errors and how to resolve them. Despite these challenges, my interest in learning and mastering Java persisted.

I encountered some difficulties in regards to the syntax of creating constructors and methods, but was able to overcome them through research and recalling the relevant concepts. I found that the key to understanding the syntax of Java was to practice and implement the concepts regularly. By doing so, I was able to remember the syntax and implement it correctly in my code. Additionally, I also took the help of online resources like tutorials and forums to understand the syntax better. Ultimately, I was able to successfully navigate and overcome these difficulties, culminating in the completion of my coursework project.

**In conclusion**, this project was a significant learning experience for me, as it allowed me to expand my skillset and adapt to new programming languages and technologies. Through my work on this project, I have developed a strong foundation in object-oriented programming concepts and techniques, particularly through the utilization of classes, objects, inheritance, and encapsulation. This project also provided me with valuable

experience in working with my instructors and teachers, and allowed me to practice and improve my report making and communication skills. I am excited to continue to develop my skills in Java programming and apply what I have learned to future projects. Despite the challenges encountered, I was able to overcome them through proper analysis, research and guidance from experts, and successfully complete the project.

## 8. APPENDIX

### 8.1 CODE OF BankCard.java

```
public class BankCard {  
  
    private int cardId;  
  
    private String clientName;  
  
    private String issuerBank;  
  
    private String bankAccount;  
  
    private int balanceAmount;  
  
    public BankCard(int balanceAmount, int cardId, String bankAccount, String  
issuerBank)  
    {  
        this.balanceAmount = balanceAmount;  
  
        this.cardId = cardId;  
  
        this.bankAccount = bankAccount;  
    }  
}
```

```
    this.issuerBank = issuerBank;

    this.clientName = "";

}

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 int getBalanceAmount()
```

```
{
```

```
    return this.balanceAmount;
```

```
}
```

```
public void setClientName(String clientName)
```

```
{
```

```
    this.clientName = clientName;
```

```
}
```

```
public void setBalanceAmount(int balanceAmount)
```

```
{
```

```
    this.balanceAmount = balanceAmount;
```

```
}
```

```
public void display()
```

```
{
```

```
    if (clientName.equals(""))
```

```
    {
```

```
        System.out.println("The Client name is not assigned.");
```

```
        System.out.println("The Card ID is: " + cardId);
```

```
        System.out.println("The Issuer Bank is: " + issuerBank);
```

```

        System.out.println("The Bank Account is: " + bankAccount);
        System.out.println("The Balance Amount is: " + balanceAmount);
    }

    else
    {
        System.out.println("The Client name is: " + clientName);
        System.out.println("The Card ID is: " + cardId);
        System.out.println("The Issuer Bank is: " + issuerBank);
        System.out.println("The Bank Account is: " + bankAccount);
        System.out.println("The Balance Amount is: " + balanceAmount);
    }
}
}
}

```

## 8.2 CODE OF DebitCard.java

```

public class DebitCard extends BankCard {

    private int pinNumber;

    private int withdrawalAmount;

    private String dateofWithdrawal;

    private boolean hasWithdrawn;
}

```

```
public DebitCard(int balanceAmount, int cardId, String bankAccount, String  
issuerBank, String clientName,int pinNumber)
```

```
{
```

```
    super(balanceAmount, cardId, bankAccount, issuerBank);
```

```
    setClientName(clientName);
```

```
    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;
}
```

```
public void setWithdrawalAmount(int withdrawalAmount)
{

    this.withdrawalAmount = withdrawalAmount;

}
```

```
public void withdraw(int withdrawalAmount, String dateofWithdrawal, int pinNumber)
{

    if (pinNumber == this.pinNumber)
    {
```



```

if (withdrawalAmount <= getBalanceAmount())
{

    setBalanceAmount(getBalanceAmount() - withdrawalAmount);

    this.withdrawalAmount = withdrawalAmount;

    this.dateofWithdrawal = dateofWithdrawal;

    this.hasWithdrawn = true;

    System.out.println("Transaction successful. Your new balance: " +
getBalanceAmount());

}

else
{

    System.out.println("Insufficient funds. You cannot withdraw ");

}

}

else
{

    System.out.println("The pin is invalid please! Enter correct pin");
}

```

```

    }
}

public void display()
{

    if (hasWithdrawn == true)
    {
        super.display();
        System.out.println("Pin Number: " + pinNumber);
        System.out.println("Withdrawal Amount: " + withdrawalAmount);
        System.out.println("Date of Withdrawal: " + dateofWithdrawal);

    }

    else
    {
        System.out.println("your Balance Amount is: " + getBalanceAmount());

    }

}

}

```

### 8.3 CODE OF CreditCard.java

```
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(int cardId, String clientName, String issuerBank, String
bankAccount, int balanceAmount,int cvcNumber, double interestRate, String
expirationDate)
    {

        super( balanceAmount, cardId, bankAccount, issuerBank);

        setClientName(clientName);

        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 String getExpirationDate()  
{  
  
    return this.expirationDate;  
  
}  
  
public int getGracePeriod()  
{  
  
    return this.gracePeriod;  
  
}  
  
public boolean getIsGranted()  
{  
  
    return this.isGranted;  
  
}
```

```
}
```

```
public void setCreditLimit(double creditLimit, int gracePeriod)
```

```
{
```

```
    if (creditLimit <= 2.5 * getBalanceAmount())
```

```
    {
```

```
        this.creditLimit = creditLimit;
```

```
        this.gracePeriod = gracePeriod;
```

```
        this.isGranted = true;
```

```
        System.out.println("Credit granted. Credit limit: " + this.creditLimit + " Grace period: " + this.gracePeriod);
```

```
    }
```

```
    else
```

```
    {
```

```
        System.out.println("you cannot grant credit limit.");
```

```
    }
```

```
}
```

```
public void cancelCreditCard()
```

```
{
```

```
    this.cvcNumber = 0;
```

```
    this.creditLimit = 0;
```

```
    this.gracePeriod = 0;
```

```
    this.isGranted = false;
```

```
}
```

```
public void display()
```

```
{
```

```
    super.display();
```

```
    if (isGranted == true)
```

```
    {
```

```
        System.out.println("Your CVC Number: " + cvcNumber);
```

```
        System.out.println("Your Credit Limit: " + creditLimit);
```

```
        System.out.println("Your Grace Period: " + gracePeriod);
```

```
        System.out.println("Your Interest Rate: " + interestRate);
```

```
        System.out.println("Your Expiration Date: " + expirationDate);
```

```
    }
```

```
else
{

    System.out.println("Your CVC Number: " + cvcNumber);
    System.out.println("Your Interest Rate: " + interestRate);
    System.out.println("Your Expiration Date: " + expirationDate);

}
}

}
```



## 9. References

harleenk\_99 & mitalibhola94, 2022. *geeksforgeeks*. [Online]  
Available at: <https://www.geeksforgeeks.org/introduction-of-bluej/>  
[Accessed 22 01 2023].

Hartman, J., 2022. *guru99*. [Online]  
Available at: <https://www.guru99.com/java-platform.html#1>  
[Accessed 25 01 2023].

Hope, C., 2020. *computerhope*. [Online]  
Available at: <https://www.computerhope.com/jargon/d/drawio.htm>  
[Accessed 24 01 2023].

Mishra, M., 2022. *scaler*. [Online]  
Available at: <https://www.scaler.com/topics/types-of-errors-in-java/>  
[Accessed 23 01 2023].