BANK ATM SOFTWARE

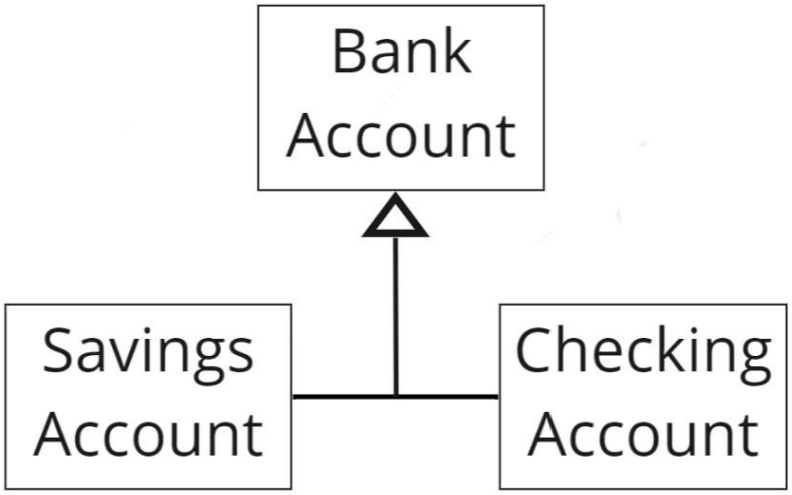
To withdraw and deposit money into various unique, personal bank accounts.

Written by William Bermel,
New York Institute of Technology

Table of Contents

Introduction

This program is designed to effectively and efficiently provide a basic user interface for clients to use a bank's ATM software. The main objective of this project is to learn firsthand the fundamentals of the business analysis of a program, including a software life cycle (SLC), create a simulated ATM featuring simple text prompts, and provide visual graphics of UML diagrams and CRC cards. Pictured below are some examples of what is provided, respectfully.



Abstract **Bank Account**	Savings Account, Checking Account
• withdraw(double amount) • deposit(double amount) • sufficientFunds(double amount) • getBalance() • getAccountNum() • getPin() • pin • accountNum • balance	• Savings Account • Checking Account • ATM

Software Life Cycle

For every piece of software in the world, there is a cycle that it follows starting with the analysis phase, which explores the problem that needs to be solved. It later moves into the designing phase, to start planning how to tackle the problem. After implementation and testing, it is deployed. Feedback is then received, which brings attention to any bugs in the code. For this software, which is a simulated ATM at a bank, the first problem that needs to be solved is to create a text based program that allows the user to enter an account number and personal ID pin number. Once that is entered, it will allow the user to deposit or withdraw money from two separate accounts, one savings account and one checking account.

Along with the software life cycle is also something called the Spiral Method. The spiral method consists of four general stages of developing a software program. The first stage is the objectives phase, where the goals are determined and laid out. The next stage is the analysis phase, where risks are identified and resolved before implementation into code. The third stage is the development and testing phase, so the software can be programmed and can be certain that it works. The last stage is to review what is currently made and to plan the next iteration of the spiral method. What can be done better? Are there any known problems? The spiral method starts out as a conceptual plan, so that everything can be put on paper and reviewed before implementation. It is then implemented, prototypes are created and then later improved on until the final result is received.

To start, I created six classes: BankAccount, SavingsAccount, CheckingAccount, ATM, User, and Bank. I then programmed the basic necessities for the BankAccount class, as well as the SavingsAccount and CheckingAccount classes, since they are very similar in skeletal composition and general functioning necessity. The withdraw and deposit functions were the first to be added, followed by the get balance method. Once these methods were functional at the most basic level, I moved on to the ATM class.

In the ATM class, I set up the majority of the user interface framework. After importing the system scanner and file reader, I set up the constructor and started writing a method called start. In this method, the main user interface would run.

Upon running, it uses an integer named consoleState in order to recognize which part of the program the user is currently using. If it is the starting menu, consoleState = 0. Each time an input is successfully accepted, consoleState changes to represent the input that was accepted.

While consoleState is 0, the user is prompted to either enter an existing account number, or type in “N” to create a new account, at which point consoleState is modified to the value of 5.

While consoleState is 5, two account numbers are automatically generated, one for the checking account and one for the savings account. While these account numbers are randomly generated, the system checks to make sure the account numbers don’t already exist. If they do, it regenerates the number. Otherwise, the user is prompted to enter a four digit pin number. At this point, both accounts have been created along with the pin number, and consoleState is modified to the value of 0, meaning that the prompt has returned to its original state. Now, the user can enter either one of the account numbers that they received before, and consoleState is modified to the value of 1.

While consoleState is 1, the menu asks for the pin that corresponds with the account number previously entered. If the pin entered is incorrect, the system will print that the pin is incorrect, and the user will be prompted to try again. This process will repeat until the user enters the correct pin that corresponds to the account number entered. Once the correct pin is entered, the user will gain access to withdraw or deposit money from the selected account.

While consoleState is 2, the user has already accessed their accounts using the account number and pin. Now, the user is presented with the options to either withdraw money from or deposit money into the selected account. They are given the option to type in either “A”, “B” or “C”, which allows them to choose to deposit, withdraw or “Exit”, respectively. After the user decides whether to withdraw or deposit, they are then prompted to enter an amount. Before the withdrawal method is called, it checks to make sure that the balance of the account has sufficient funds in order to withdraw the amount requested. Once the amount

has been entered, it will display the new balance of the selected account, and the same prompt from before is presented again, allowing the user to either withdraw, deposit, or “Exit”.

The user can deposit as much money as they want, and withdraw as much money as is in the account, and this process can repeat for however many times the user chooses to do so, until the user chooses the “Exit” option. If the user chooses to “Exit” at any point during the program, consoleState will be changed to the value of 0, which means the user will have to reenter their account number, or create a new set of accounts and pin.

Everything using consoleState is the main, front-end part of the program, meaning it’s the only thing that the user will see. At this point in the development of the program in its entirety, I tried to start implementing writing the account numbers and pin numbers to a file, so that it is more secure and also to showcase my knowledge of the subject. However, after writing the code to write all of the accounts into a txt file, I kept receiving many errors for different reasons. I decided that this was a problem best slept on, so I tried to tackle it later.

In the meantime, I chose to write all the other methods needed that the user would not directly see. This included the withdrawal and deposit methods in the BankAccount class, which were inherited by the SavingsAccount and CheckingAccount classes, as well as some getter methods and a method titled sufficientFunds that would return true if the balance of the given account was greater than or equal to a given amount trying to be withdrawn.

Also, I wrote the User class, which only consists of getter methods. The main purpose of having these methods in this class is to allow the ATM class to easily access the information it needs in order to run.

At this point I decided to switch back to the ATM class, where I was previously presented with the issue of not being able to write to the desired file. After some attention to detail, I overcame this obstacle and continued to test the rest of the program (see Troubleshooting section). While testing, I was able to find yet

another complication with the code. This new issue, which was that the reader was unable to find the account number in the file, was nothing major to worry about, but it was still preventing the program from running the way it was expected to run.

I decided that the best course of action would be to keep testing until this error was resolved, because otherwise the rest of the code after it would be untestable, and testing all of the code was the highest priority for me. After fixing the error in the code, I was able to test all of the code written in the ATM class, which was the foundation of the program's text based user interface.

Another small error presented itself to me, but after some logical analysis, it was soon resolved. Now, at this point, the entirety of the code had been written, tested and the known bugs had been squashed.

CRC Cards

ATM	
• start() • addAccount(int accNum, int pinNum) • checkPin(int enteredAccNum, int enteredPinNum) • accounts • currentAccNum • currentAccount • in • reader • writer • quit • consoleState • f • client	• Bank • BankAccount • User

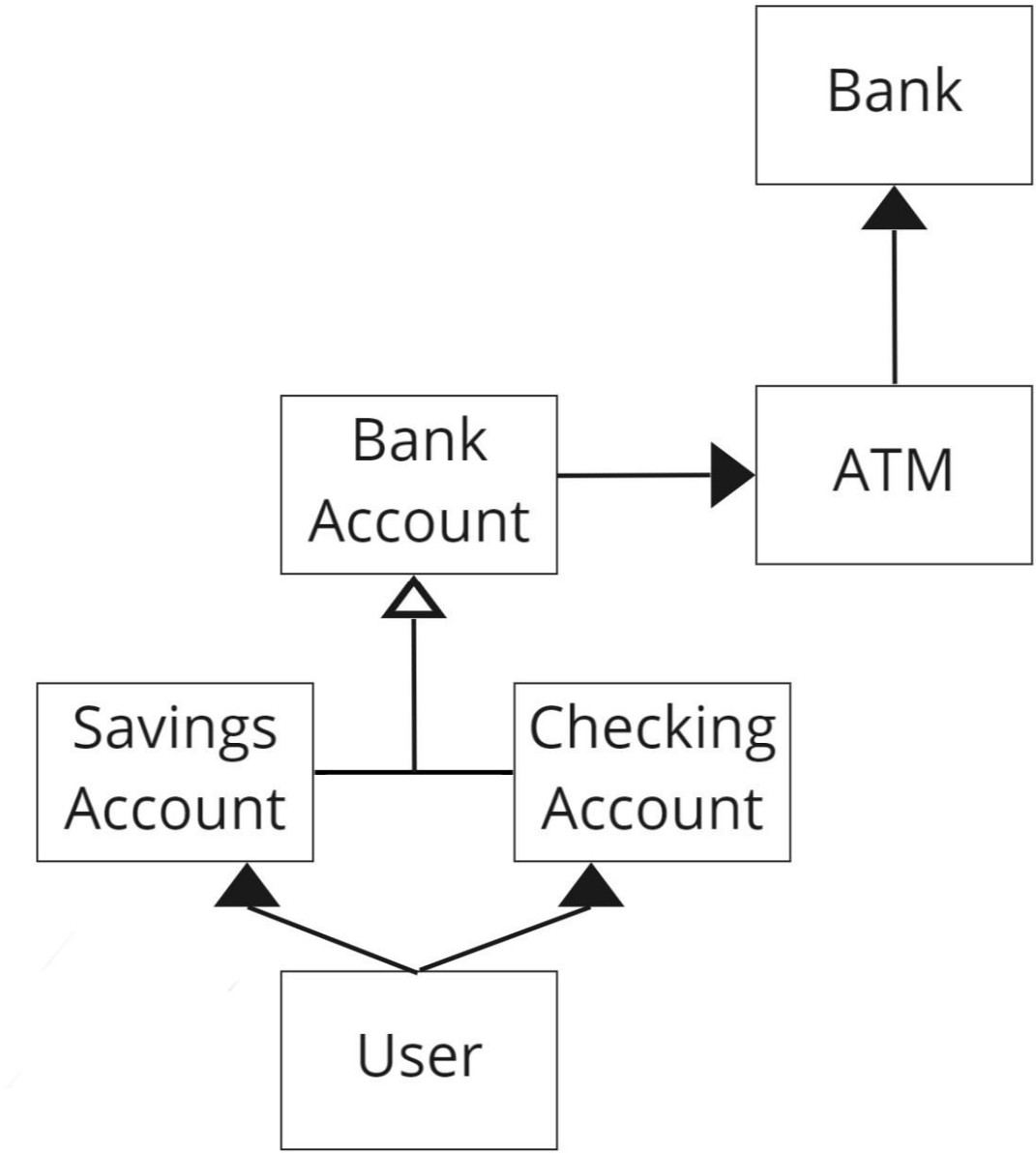
Abstract Bank Account	Savings Account, Checking Account
• withdraw(double amount) • deposit(double amount) • sufficientFunds(double amount) • getBalance() • getAccountNum() • getPin() • pin • accountNum • balance	• Savings Account • Checking Account • ATM

CheckingAccount	BankAccount
• withdraw(double amount) • deposit(double amount) • getBalance() • getAccountNum()	

SavingsAccount	BankAccount
• withdraw(double amount) • deposit(double amount) • getBalance() • getAccountNum()	

User	
• deposit(BankAccount account, double amount) • withdraw(BankAccount account, double amount) • getChecking(int accNum) • getSavings(int accNum) • getType(int accNum) • getBalance(BankAccount account) • setCheckingAccount(int accountNum) • setSavingsAccount(int accountNum) • setPin(int newPin) • pin • checkingAccount • savingsAccount	• ATM • BankAccount • CheckingAccount • SavingsAccount

UML Diagram


Bank
Bank Account
ATM
Savings Account
Checking Account
User

Method Documentation

```
/**
	Connects the user to the bank accounts
*/
public class ATM{
/**
	Runs the main code required for the user to interact with the program.
*/
public void start(){
}
/**
	Adds a bank account to the file
	@param accNum   randomly generated account number
	@param pinNum   user entered pin number
*/
public void addAccount(int accNum, int pinNum) throws IOException{
}
/**
	Checks the entered pin number against the pin on file
	@param enteredAccNum          user entered account number
	@param enteredPinNum user entered pin number
*/
public boolean checkPin(int enteredAccNum, int enteredPinNum){
}
}

/**
	Client that contains a pin, checking account, and savings account.
*/
public class User{
/**
	Deposits an amount of money to the requested account
	@param account    bank account being accessed
```

```
        @param amount    user entered amount of money
*/
public void deposit(BankAccount account, double amount){
}
/**
        Withdraws an amount of money to the requested account
        @param account    bank account being accessed
        @param amount    user entered amount of money
*/
public void withdraw(BankAccount account, double amount){
}
/**
        Returns a checking account given an account number
        @param accNum   bank account being accessed
*/
public CheckingAccount getChecking(int accNum) {
}
/**
        Returns a savings account given an account number
        @param accNum   bank account number being entered
*/
public SavingsAccount getSavings(int accNum) {
}
/**
        Returns a bank account given an account number
        @param accNum   bank account number being entered
*/
public BankAccount getType(int accNum) {
}
/**
        Returns the balance of an account given an account number
        @param accNum   bank account being accessed
*/
public double getBalance(BankAccount account) {
```

```
}
/**
        Designates a checking account to contain an account number
        @param accNum   bank account number being entered
*/
public void setCheckingAccount(int accountNum) {
}

/**
        Designates a savings account to contain an account number
        @param accNum   bank account number being entered
*/
public void setSavingsAccount(int accountNum) {
}
/**
        Designates a pin number for the user's accounts
        @param pinNum   user's desired pin number
*/
public void setPin(int newPin) {
}
}
/**
        Account that withdraws and deposits, amongst other actions
*/
public abstract class BankAccount {
/**
        Deposits amount from the bank account
        @param amount    amount being deposited
*/
public void deposit(double amount) {
}
/**
        Withdraws amount from the bank account
        @param amount    amount being withdrawn
```

```
*/
public void withdraw(double amount) {
}
/**
        Checks that the amount being withdrawn is not greater
        than the balance of the account
        @param amount    amount being withdrawn
*/
public boolean sufficientFunds(double amount) {
}
/**
        Returns balance of the bank account
*/
public double getBalance() {
}


/**
        Returns account number of the bank account
*/
public int getAccountNum() {
}
/**
        Returns pin number of the bank account
*/
public int getPin() {
}
/**
        Savings account that withdraws and deposits, amongst other actions
*/
public class SavingsAccount extends BankAccount{
/**
        Deposits amount from the bank account
        @param amount    amount being deposited
```

```
*/
public void deposit(double amount) {
	super.deposit(amount);
}
/**
	Withdraws amount from the bank account
	@param amount	amount being withdrawn
*/
public void withdraw(double amount) {
}



/**
	Returns the balance of the account
*/
public double getBalance() {
}
/**
	Returns the account number of the account
*/
public int getAccountNum() {
}
}
/**
	Checking account that withdraws and deposits, amongst other actions
*/
public class CheckingAccount extends BankAccount{
/**
	Deposits amount from the bank account
	@param amount	amount being deposited
*/
public void deposit(double amount) {
}
```

```
/**
        Withdraws amount from the bank account
        @param amount    amount being withdrawn
*/
public void withdraw(double amount) {
}



/**
        Returns the balance of the account
*/
public double getBalance() {
}
/**
        Returns the account number of the account
*/
public int getAccountNum() {
}
}
```

Implementation

Now, we can implement the constructors, instance fields, and the bodies of the methods.

```
public class CheckingAccount extends BankAccount{
	public CheckingAccount() {
		super();
	}
	public CheckingAccount(double bal, int accNum, int pinNum) {
		super(bal, accNum, pinNum);
	}

	public void deposit(double amount) {
		super.deposit(amount);
	}
	public void withdraw(double amount) {
		super.withdraw(amount);
	}

	public double getBalance() {
		return super.getBalance();
	}
	public int getAccountNum() {
		return super.getAccountNum();
	}
}



public class SavingsAccount extends BankAccount{
	public SavingsAccount() {
		super();
	}
	public SavingsAccount(double bal, int accNum, int pinNum) {
```

```
            super(bal, accNum, pinNum);
        }

        public void deposit(double amount) {
            super.deposit(amount);
        }
        public void withdraw(double amount) {
            super.withdraw(amount);
        }

        public double getBalance() {
            return super.getBalance();
        }
        public int getAccountNum() {
            return super.getAccountNum();
        }
}

public abstract class BankAccount {
        public BankAccount() {
                    balance = 0;
                    accountNum = 0;
                    pin = 0;
        }

        public BankAccount(double bal, int accNum, int pinNum) {
            balance = bal;
            accountNum = accNum;
            pin = pinNum;
        }

        public void deposit(double amount) {
            try {
                    System.out.print("Depositing: $" + amount);
```

```
                double newBalance = balance + amount;
                System.out.println(". new balance is: " + newBalance);

                balance = newBalance;
        }
        catch(Exception e) {
                e.printStackTrace();
        }
}

public void withdraw(double amount) {
        try {
                System.out.print("Withdrawing: $" + amount);
                double newBalance = balance - amount;
                System.out.println(". New balance is: " + newBalance);
                balance = newBalance;
        }
        catch(Exception e){
                e.printStackTrace();
        }
}

public boolean sufficientFunds(double amount) {
        if(this.balance - amount >= 0)
                return true;
        return false;
}

public double getBalance() {
        return this.balance;
}
public int getAccountNum() {
        return this.accountNum;
}
```

```
	public int getPin() {
		return this.pin;
	}

	private int pin;
	private int accountNum;
	private double balance;
}
```

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

public class ATM {

	public ATM(User anUser) throws FileNotFoundException, IOException {
		client = anUser;
		f = new File("BankClients.txt");
		if(!f.exists())
			f.createNewFile();
		in = new Scanner(System.in);
		consoleState = 0;
		quit = false;
		accounts = new ArrayList<String>();
	}
```

```
	public void start() throws IOException {


		try {
			while(quit == false) {
				while(consoleState == 0) {
					reader = new BufferedReader(new FileReader(f));
					System.out.print("Enter account number or enter N
to create a new account: ");
					String str = in.next();
					if(str.equalsIgnoreCase("N")) {
						consoleState = 5;
						break;
					}
					String line = "";
					currentAccNum = Integer.parseInt(str);
					while(true) {
						if((line = reader.readLine()) != null) {
							if(line.substring(0, 5).equals(str)) {
								consoleState = 1;
								break;
							}
						} else{
							System.out.print("Invalid account
number. Please try again. \n");
							break;
						}

					}

				}

				reader.close();
				while(consoleState == 1) {
```

```
                        System.out.print("\nPlease enter pin number: ");
                        int newPin = in.nextInt();
                        if(checkPin(currentAccNum, newPin)) {
                                System.out.print("Pin accepted.");
                                consoleState = 2;
                        } else
                                System.out.print("Invalid pin number,
please try again. \n");
                }

                while(consoleState == 2) {
                        System.out.println();

        if(client.getType(currentAccNum).equals(client.getChecking(currentAccNu
m))) {
                                System.out.print("Balance of checking
account: " + client.getType(currentAccNum).getBalance());
                                this.currentAccount =
client.getChecking(currentAccNum);

                        } else {
                                System.out.println();
                                System.out.print("Balance of savings
account: " + client.getType(currentAccNum).getBalance());
                                this.currentAccount =
client.getSavings(currentAccNum);

                        }

                        System.out.println();
                        System.out.print("Please select an option: A =
Withdraw  B = Deposit  C = Exit \n");
                        String entrant = in.next();
                        if(entrant.equalsIgnoreCase("C")) {
```

```
                                        System.out.print("Returning to the main
menu.");
                                        System.out.print("\n\n\n");
                                        consoleState = 0;
                                } else if(entrant.equalsIgnoreCase("A")) {
                                        System.out.println();
                                        System.out.print("How much would you
like to withdraw?");
                                        double tempWithdrawal = in.nextDouble();

        if(this.currentAccount.sufficientFunds(tempWithdrawal))
                                                client.withdraw(this.currentAccount,
tempWithdrawal);
                                        else
                                                System.out.print("Insufficient
funds");

                                } else if(entrant.equalsIgnoreCase("B")) {
                                        System.out.println();
                                        System.out.print("How much would you
like to deposit?");
                                        double tempDeposit = in.nextDouble();
                                        client.deposit(this.currentAccount,
tempDeposit);
                                } else {
                                        System.out.print("Invalid input. Please try
again.");
                                }
                        }

                        while(consoleState == 5) {
                                int newAcc;
                                System.out.println();
```

```
                                System.out.print("Generating bank account
number... \n");

                                System.out.print("Enter desired 4 digit pin
number: ");
                                int newPin = in.nextInt();
                                client.setPin(newPin);
                                newAcc = (int) ((Math.random() * 89999) +
10000);
                                while(accounts.contains("" + newAcc))
                                        newAcc = (int) ((Math.random() * 89999) +
10000);

                                System.out.print("Checking account number: " +
newAcc + "\n");
                                addAccount(newAcc, newPin);
                                client.setCheckingAccount(newAcc);

                                newAcc = (int) ((Math.random() * 99999) +
10000);
                                while(accounts.contains("" + newAcc))
                                        newAcc = (int) ((Math.random() * 99999) +
10000);

                                System.out.print("Savings account number: " +
newAcc + "\n");
                                addAccount(newAcc, newPin);
                                client.setSavingsAccount(newAcc);

                                consoleState = 0;
                        }
                }

        } catch (IOException e) {
```

```
            e.printStackTrace();
        } finally {
            in.close();
            reader.close();
            writer.close();
        }
    }



    public void addAccount(int accNum, int pinNum) throws IOException {

        try {
            String temp = ("" + accNum + " " + pinNum + "\n");
            String accountList = "";
            String str;
            reader = new BufferedReader(new FileReader(f));
            while((str = reader.readLine()) != null)
                accountList += str + "\n";

            accountList += temp;
            reader.close();
            writer = new FileWriter(f);
            writer.write(accountList);

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            reader.close();
            writer.close();
        }
    }

    public boolean checkPin(int enteredAccNum, int enteredPinNum) {
```

```
            if(client.getType(enteredAccNum).getPin() == enteredPinNum)
                return true;
            return false;
        }

        private ArrayList<String> accounts;
        private int currentAccNum;
        private BankAccount currentAccount;
        private Scanner in;
        private BufferedReader reader;
        private FileWriter writer;
        private boolean quit;
        private int consoleState;
        private File f;
        private User client;
}

public class User {
        public User() {
            pin = 0;
            checkingAccount = new CheckingAccount();
            savingsAccount = new SavingsAccount();
        }

        public void deposit(BankAccount account, double amount) {
            account.deposit(amount);
        }
        public void withdraw(BankAccount account, double amount) {
            account.withdraw(amount);
        }

        public CheckingAccount getChecking(int accNum) {
            if(accNum == this.checkingAccount.getAccountNum())
                return this.checkingAccount;
```

```
            else
                return null;
        }
        public SavingsAccount getSavings(int accNum) {
            if(accNum == this.savingsAccount.getAccountNum())
                return this.savingsAccount;
            else
                return null;
        }
        public BankAccount getType(int accNum) {
            if(accNum == this.savingsAccount.getAccountNum())
                return this.savingsAccount;
            else
                return this.checkingAccount;
        }
        public double getBalance(BankAccount account) {
            return account.getBalance();
        }

        public void setCheckingAccount(int accountNum) {
            this.checkingAccount = new CheckingAccount(0, accountNum, pin);
        }
        public void setSavingsAccount(int accountNum) {
            this.savingsAccount = new SavingsAccount(0, accountNum, pin);
        }
        public void setPin(int newPin) {
            this.pin = newPin;
        }



        private int pin;
        private CheckingAccount checkingAccount;
        private SavingsAccount savingsAccount;
}
```

Actual Code (Java)

```
48              reader.close();
49              while(consoleState == 1) {
50                  System.out.print("\nPlease enter pin number: ");
51                  int newPin = in.nextInt();
52                  if(checkPin(currentAccNum, newPin)) {
53                      System.out.print("Pin accepted.");
54                      consoleState = 2;
55                  } else
56                      System.out.print("Invalid pin number, please try again. \n");
57              }
58
59              while(consoleState == 2) {
60                  System.out.println();
61                  if(client.getType(currentAccNum).equals(client.getChecking(currentAccNum))) {
62                      System.out.print("Balance of checking account: " + client.getType(currentAccNum).getBalance());
63                      this.currentAccount = client.getChecking(currentAccNum);
64
65                  } else {
66                      System.out.println();
67                      System.out.print("Balance of savings account: " + client.getType(currentAccNum).getBalance());
68                      this.currentAccount = client.getSavings(currentAccNum);
69
70                  }
71
72                  System.out.println();
73                  System.out.print("Please select an option: A = Withdraw  B = Deposit  C = Exit \n");
74                  String entrant = in.next();
75                  if(entrant.equalsIgnoreCase("C")) {
76                      System.out.print("Returning to the main menu.");
77                      System.out.print("\n\n\n");
78                      consoleState = 0;
79                  } else if(entrant.equalsIgnoreCase("A")) {
80                      System.out.println();
81                      System.out.print("How much would you like to withdraw?");
82                      double tempWithdrawal = in.nextDouble();
83                      if(this.currentAccount.sufficientFunds(tempWithdrawal))
84                          client.withdraw(this.currentAccount, tempWithdrawal);
85                      else
86                          System.out.print("Insufficient funds");
87
88                  } else if(entrant.equalsIgnoreCase("B")) {
89                      System.out.println();
90                      System.out.print("How much would you like to deposit?");
91                      double tempDeposit = in.nextDouble();
92                      client.deposit(this.currentAccount, tempDeposit);
93                  } else {
94                      System.out.print("Invalid input. Please try again.");
```

```
93                  } else {
94                      System.out.print("Invalid input. Please try again.");
95                  }
96              }
97
98              while(consoleState == 5) {
99                  int newAcc;
100                 System.out.println();
101                 System.out.print("Generating bank account number... \n");
102
103                 System.out.print("Enter desired 4 digit pin number: ");
104                 int newPin = in.nextInt();
105                 client.setPin(newPin);
106                 newAcc = (int) ((Math.random() * 89999) + 10000);
107                 while(accounts.contains("" + newAcc))
108                     newAcc = (int) ((Math.random() * 89999) + 10000);
109
110                 System.out.print("Checking account number: " + newAcc + "\n");
111                 addAccount(newAcc, newPin);
112                 client.setCheckingAccount(newAcc);
113
114                 newAcc = (int) ((Math.random() * 99999) + 10000);
115                 while(accounts.contains("" + newAcc))
116                     newAcc = (int) ((Math.random() * 99999) + 10000);
117
118                 System.out.print("Savings account number: " + newAcc + "\n");
119                 addAccount(newAcc, newPin);
120                 client.setSavingsAccount(newAcc);
121
122                 consoleState = 0;
123             }
124         }
125
126     } catch (IOException e) {
127         e.printStackTrace();
128     } finally {
129         in.close();
130         reader.close();
131         writer.close();
132     }
133 }
134
```

```
137     public void addAccount(int accNum, int pinNum) throws IOException {
138 
139         try {
140             String temp = ("" + accNum + " " + pinNum + "\n");
141             String accountList = "";
142             String str;
143             reader = new BufferedReader(new FileReader(f));
144             while((str = reader.readLine()) != null)
145                 accountList += str + "\n";
146 
147             accountList += temp;
148             reader.close();
149             writer = new FileWriter(f);
150             writer.write(accountList);
151 
152         } catch (IOException e) {
153             e.printStackTrace();
154         } finally {
155             reader.close();
156             writer.close();
157         }
158     }
159 
160     public boolean checkPin(int enteredAccNum, int enteredPinNum) {
161         if(client.getType(enteredAccNum).getPin() == enteredPinNum)
162             return true;
163         return false;
164     }
165 
166     private ArrayList<String> accounts;
167     private int currentAccNum;
168     private BankAccount currentAccount;
169     private Scanner in;
170     private BufferedReader reader;
171     private FileWriter writer;
172     private boolean quit;
173     private int consoleState;
174     private File f;
175     private User client;
176 }
177 
```

```
1
2  public abstract class BankAccount {
3      public BankAccount() {
4              balance = 0;
5              accountNum = 0;
6              pin = 0;
7      }
8      public BankAccount(double bal, int accNum, int pinNum) {
9          balance = bal;
10         accountNum = accNum;
11         pin = pinNum;
12     }
13
14     public void deposit(double amount) {
15         try {
16             System.out.print("Depositing: $" + amount);
17             double newBalance = balance + amount;
18             System.out.println(". new balance is: " + newBalance);
19
20             balance = newBalance;
21         }
22         catch(Exception e) {
23             e.printStackTrace();
24         }
25     }
26
```

```
27    public void withdraw(double amount) {
28        try {
29            System.out.print("Withdrawing: $" + amount);
30            double newBalance = balance - amount;
31            System.out.println(". New balance is: " + newBalance);
32            balance = newBalance;
33        }
34        catch(Exception e){
35            e.printStackTrace();
36        }
37    }
38
39    public boolean sufficientFunds(double amount) {
40        if(this.balance - amount >= 0)
41            return true;
42        return false;
43    }
44
45    public double getBalance() {
46        return this.balance;
47    }
48    public int getAccountNum() {
49        return this.accountNum;
50    }
51    public int getPin() {
52        return this.pin;
53    }
54
55    private int pin;
56    private int accountNum;
57    private double balance;
58 }
59
```

```
1
2  public class CheckingAccount extends BankAccount{
3      public CheckingAccount() {
4          super();
5      }
6      public CheckingAccount(double bal, int accNum, int pinNum) {
7          super(bal, accNum, pinNum);
8      }
9
10     public void deposit(double amount) {
11         super.deposit(amount);
12     }
13     public void withdraw(double amount) {
14         super.withdraw(amount);
15     }
16
17     public double getBalance() {
18         return super.getBalance();
19     }
20     public int getAccountNum() {
21         return super.getAccountNum();
22     }
23
24 }
```

```java

public class SavingsAccount extends BankAccount{
    public SavingsAccount() {
        super();
    }
    public SavingsAccount(double bal, int accNum, int pinNum) {
        super(bal, accNum, pinNum);
    }

    public void deposit(double amount) {
        super.deposit(amount);
    }
    public void withdraw(double amount) {
        super.withdraw(amount);
    }

    public double getBalance() {
        return super.getBalance();
    }
    public int getAccountNum() {
        return super.getAccountNum();
    }


}
```

```
1
2  public class User {
3      public User() {
4          pin = 0;
5          checkingAccount = new CheckingAccount();
6          savingsAccount = new SavingsAccount();
7      }
8
9      public void deposit(BankAccount account, double amount) {
10         account.deposit(amount);
11     }
12     public void withdraw(BankAccount account, double amount) {
13         account.withdraw(amount);
14     }
15
16     public CheckingAccount getChecking(int accNum) {
17         if(accNum == this.checkingAccount.getAccountNum())
18             return this.checkingAccount;
19         else
20             return null;
21     }
22     public SavingsAccount getSavings(int accNum) {
23         if(accNum == this.savingsAccount.getAccountNum())
24             return this.savingsAccount;
25         else
26             return null;
27     }
```

```
28      public BankAccount getType(int accNum) {
29          if(accNum == this.savingsAccount.getAccountNum())
30              return this.savingsAccount;
31          else
32              return this.checkingAccount;
33      }
34      public double getBalance(BankAccount account) {
35          return account.getBalance();
36      }
37
38      public void setCheckingAccount(int accountNum) {
39          this.checkingAccount = new CheckingAccount(0, accountNum, pin);
40      }
41      public void setSavingsAccount(int accountNum) {
42          this.savingsAccount = new SavingsAccount(0, accountNum, pin);
43      }
44      public void setPin(int newPin) {
45          this.pin = newPin;
46      }
47
48
49      private int pin;
50      private CheckingAccount checkingAccount;
51      private SavingsAccount savingsAccount;
52  }
53
```

Conclusion

In all, this program is designed to allow any user to follow a text based interface and create their own accounts with randomly generated account numbers and a pin number that they input. With these accounts they are able to withdraw and deposit money, or exit back to the main menu. Although there were some logical issues with the code, they were later resolved and naturally more issues arose! But, after all of the problems being fixed, the program became fully functional. This report is also designed to show each step of the process of the creation of the program.

Troubleshooting

Like any software program, there were many coding errors that were encountered during the production of the program. First, the main obstacle that was confronted was being unable to read from and write to the desired txt file. After some trial and error, I was able to figure out that the main problem was the location of where the file reader and file writer was being initialized and then closed.

The next problem that I came across was an exception being thrown because an amount would be withdrawn from the account that was greater than the balance of the account. This was a simple fix, as I simply created a new method titled sufficientFunds that would return a boolean and check to make sure that the amount being withdrawn was less than or equal to the balance of the account. Then, the user would only be able to withdraw from the account after the condition had been met.

Another minor problem that was later discovered was that certain inputs could not be read unless they were repeated a certain number of times. For example, if the user wished to exit the program, they would have to enter “C” three times until it worked, and the same thing for depositing and entering “B”. The solution to this problem was to have the scanner take the next input as a string before checking whether it was equal to the letters required for the code to work. Previously, it would check the next string inside of the if statement, and the program would need another input before it moved on to the next condition.