Partner Bank Account CS Assignment

Javin, Caleb, Mark

IMPORT
java.util.*;
java.io.*;

METHOD 1 - New Account {newAccount}
Creates a new account into a file

{Parameter(s)} ~

- Gives random account number with 6 digits
- Creates a new file for the account
- Asks for user first name
- Asks for user last name
- Gets pin
- Confirms Pin with an if statement
- Adds all data into the customers file that can be accessed later to log into account

METHOD 2 - Check Account (see if it exists) {checkDetails}

{Parameter(s)} ~

- Runs through the previously created files in New Account to see if account is valid
  - Uses while loop to continuously run through the files
- Compares account number first, if it passes; checks pin
- Makes boolean true to be able to continue with the account options
- If the account isn't valid, prints an error statement and allows the user to try again or create a new account

METHOD 3 - INTERIM METHOD TO CREATE OPTION SCREEN (***no longer used, put in main method***)

{Parameter(s)} ~ accNumber - uses account

- Asks user if they want to log into their account or open a new account
  - Asks user if they want to check balance, deposit, withdraw, close their account
    - Gets user input
    - Uses if statement to go to method for desired option
      - If there is no option with that name, prints error message and asks again

METHOD 4 - Check Account Balance {checkAccBalence}

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

  - Opens up user account file
    - Uses while statement to sort through all saved files
  - Reads current balance in the account
  - Prints out current balance into console alongside the account name

METHOD 5 - Deposit Money {depositMoney}

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

- Opens users file from login stage
- Asks user which account number they would like to deposit to
  - Finds account using loop, and opens that account file
  - Prints error message if there is no account with that name, asks again

- Asks for pin number
    - If pin is correct
        - Asks user how much money they would like to deposit
        - Reads the file to see how much money is in their account
        - That number is declared to a where the deposited number is added to the balance.
        - Uses and array to read the current balance and get rid of it
        - writes and prints the new current balance
    - If pin is incorrect
        - Prints error message “invalid pin”, asks again

METHOD 6 - Withdraw Money {withdrawMoney}

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

- Opens users file from login stage
- Asks which account they would like to open
    - While loop loops through their account to find either sav or checkings account
        - If they don't have the account they tried to draw from, print error message and ask for another account
- If the account exists, asks the user how much money they want to withdraw
    - If statement to check if they have that much in their account
- If the if statement is valid it subtracts the money from the account, and ‘prints’ it out
    - If the if statement isn’t valid, it prints an error message and doesn’t print any money out
- Returns to main print out screen to access another action

METHOD 7 - Close Account {closeAccount}

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

- Opens users file from login stage
- Asks user which account they would like to delete
- Gets user to reenter their pin
- Double checks that the user actually want to delete your account
    - Tells user how much money they will lose if they close the account
        - Triple checks that they really want to delete the account
- Deletes the specified account

METHOD 8 - Open Account

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

- Opens users file
- Asks which account they would like to open (checking/savings)
    - Uses if statement to open either account
- Writes “chequing” or “saving” on the file, and starting balance at 0

METHOD 9 - Change Pin {changePin}

{Parameter(s)} ~ accNumber

- Ask user to reenter their pin
- If valid, ask for the users new pin number
- Change the pin number to the new pin number

METHOD 10 - Return To Login Screen/Log out

{Parameter(s)} ~ accNumber - uses account number from login stage to get the file from system

- Says “returning ” + *accNumber*

- Returns the program to login screen

METHOD MAIN - Gets Account Number, Pin and loads Main Screen

{Parameter(s)} ~ (VOID **main***)*

- Use infinite loop to ask user if they want to log into their account, open a new account, or end the program
- Ask user for the account number
- Ask user for the pin to the specified account number
    - Displays options for user (deposit, withdraw, etc)
        - Gets user input
        - Uses if statement to go to method for desired option
            - If there is no option with that name, prints error message and asks again

For this assignment, you and a partner will act as software engineers AND developers.
You will go through each of the stages in the software development process to ensure that your software is correctly designed before you begin coding, and then thoroughly tested before it is completed.

Your software should follow all good coding style practices, and should have input validation loops any time the user is allowed to enter input.

The following document will detail your expectations for each stage of this assignment.

There is a **HARD DEADLINE** for this assignment on Tuesday, May 31st. If you are not complete, you should submit whatever you have done, along with a document detailing what parts of your system are **not working properly**, or **missing**.

Requirements stage:

You will not have to worry about the requirements stage, as this document will effectively serve as your project requirements document. Your client (i.e., Mr. Skuja) has reuested that you design a banking ATM system. Your system must have the following features:

- It must store the information of the bank's customers in a saved file.
    - Each customer should have a 6-digit customer number, a first & last name, and a 4-digit personal identification number (PIN) which works as a password.
    - Each customer may also have a checking account, a savings account, or both. Your system should know which kinds of accounts a customer has, and how much money they have in each account.

- When your system begins, it should prompt the user to enter their customer number and PIN.
    - If they enter a valid customer number but an incorrect PIN, your program should give an error message.
    - If they do not enter a valid customer number, your software should print a message stating that the customer does not exist. They should be given the choice of creating a new customer profile with the PIN given, or trying again.
        - If a new customer profile is created, the user should provide their first & last name.
    - If they are successful, they should be given access to a menu of options.

    - Once the user finishes, this login screen should **loop infinitely**. The device your software is intended to run on will be running constantly, and will be powered off when the software is to be shut down.

- Once a user is given access to the system, they should be able to use a menu system to do any of the following:
    - Checking their account balance(s)
    - Depositing money into an account
        - If they have more than one account, they should be able to choose which one they deposit to.
        - The system should ask the user how much they want to deposit, then update their account and print out their new balance.
    - Withdrawing money from their account
        - If they have more than one account, they should be able to choose which one they deposit to
        - The system should ask the user how much they want to withdraw, then update their account and print out their new balance.
    - Closing an account
        - The system should print out how much money was in the account before closing it, and then the account should be removed from the customer profile.
    - Open a new account
        - If the user has no accounts, it should ask what kind they would like to create, and set the balance to $0.00
        - If the user only has one type of account, the other type of account should be created with a balance of $0.00
        - If the user has both types of accounts, they should receive an error message that they already have both types.
    - Change their PIN number
    - Log out (return to the user login loop)

- Any time the user makes a change to their customer profile – whether opening or closing an account, depositing or withdrawing funds, or changing their password – their details should be saved.

- Your system should ensure that any actions performed make sense (for example, a user should not be able to withdraw more money than an account has) and should make of input validation loops whenever input is allowed. It should not be possible to crash the system.

Design stage:

During your design stage, you should be creating a document that outlines:

- The control flow for your main method (How does it run? Where are the loops? What decisions get made? When are methods run? When will information be loaded? When will it be saved?)

- Each of the methods in your system (What parameters should they take in? What will they do? What will they return?)

- How each of the files in your system will be designed (In what way will the information be saved so you know how to load it later?)

Your document should detail a complete idea of how your system will function before you move on from this stage. **You should meet with me before continuing past this stage.**

Implementation stage:

During this stage you will code your system. There shouldn't be any surprises here.

As you work on this stage, you will be expected to keep a **daily activity log** for you and your partner, detailing what you worked on each day. This will be part of your submission.

Verification stage:

As you complete each method, you should test it independently to ensure it works properly.

Similarly, you should test your system whenever you introduce any part of your system to another part of your system (such as when something you have worked on now connects to a method your partner has created, or when you start using one of your methods as part of main()).

If problems are encountered, you should return to the implementation stage. You will be expected to go back and forth between these two stages until your program is stable.

Remember that you should also be testing your program at regular times to see if you can "break" it while it is running by entering bad data.

Time spent in the verification stage can also be logged in your **daily activity log**!

Maintenance stage:

You will also not be responsible for this stage. Once you have completed verification, your task here is done!

Rubrics:

Thinking & Inquiry – Design document

Level 0	Most of the system requirements have not been detailed in the design document.
Level 1	The design document is missing significant elements of the system requirements.
Level 2	The vast majority of the system has been designed, though some requirements are not covered **OR** Some of the methods in the system are not adequately designed. **OR** Some parts of the file system are not adequately planned out.
Level 3	The students have designed the entirety of the system, covering all functionality from the system requirements, file formats, and methods where necessary.
Level 4	Meets all the expectations of level 3, but also does so in a fashion where all aspects of the of the system are very well defined, giving a very clear understanding of how the system should be implemented.

Application – Implementation & Verification

Level 0	System implementation has not begun.
Level 1	System implementation is somewhat underway. **OR** Activity logs are not present
Level 2	System implementation is well underway, but system is not functional. **OR** Daily activity logs are not sufficiently detailed or have significant gaps
Level 3	The user login process & main menu have been implemented, along with the main menu input & processing loop, and some options from the main menu are functional. Daily activity logs for all partners are detailed and demonstrate an adequate use of development time.
Level 4	Meets the expectations of level 3, but most (though not necessarily all) options are functional.

Communication – Coding Style & Documentation

Level 0	System is missing comments, program header comment blocks & method header comment blocks **OR** Code is consistently demonstrating multiple poor coding practices
Level 1	System shows some level of commenting, but significant portions are missing commenting **OR** Code demonstrates noticeable amounts of poor coding practices
Level 2	System is mostly commented appropriately, though there are still noticeable absences **OR** Code demonstrates some poor coding practices
Level 3	System is properly commented, but is lacking in appropriate level of detail or sufficient description **OR** Code demonstrates very few poor coding practices
Level 4	System is fully commented, and all comments are appropriately thorough **AND** Code properly follows all rules of good coding style

CODED METHODS

GLOBAL VARIABLES

```
  public static int newAccountFileName = 000001;
  public static String convertAccFileName = "";
  public static boolean accountCheck = false;
  public static double randNums = 0;
  public static int randAccNum = 0;
```

METHOD 1

METHOD 2

METHOD 3

METHOD 4

METHOD 5

METHOD 6

METHOD 7

METHOD 8

METHOD 9

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

/*
BankAccount
Javin Liu
Mark Ungar
Caleb Yoon
2022-05-23
This program creates individual ID codes and pins that is fully interactable as a bank account
*/

public class BankAccount {

  public static String loginAccNum = "";//declares/initilizes variables
  public static int validAccount;//declares/initilizes variables

  /*
  newAccount(int accNumCheck, int accPinCheck)
  returns accNumCheck
  int accNumCheck - checks if the account num is valid, int accPinCheck - checks if the pin num is valid
  This method checks if someones account is valid
  */

  public static int newAccount (int accNumCheck, int accPinCheck) {
    Scanner sc = new Scanner(System.in);//scanner

    int convertRandNums, userPinNum, userPinNumCheck, fourDigitCheck;//declares/initilizes variables
    String firstName = "", lastName = "", formattedAccNum = "";//declares/initilizes variables

    double accNum;//declares/initilizes variables
    int intAccNum;//declares/initilizes variables


    try
    {
      //declares/initilizes variables
      accNum = Math.random();
      accNum = accNum * 1000000;
      intAccNum = (int)accNum;
      formattedAccNum = Integer.toString(intAccNum);

      //prints users account number
      System.out.println("Your new account number is " + intAccNum);
```

```
        BufferedWriter out = new BufferedWriter(new FileWriter(formattedAccNum, false));//opens writer

        //asks user for their 4 digit pin, user inputs
        System.out.print("Please enter a four digit pin: ");
        userPinNum = sc.nextInt();

        fourDigitCheck = String.valueOf(userPinNum).length();//makes sure it is 4 digits

        //if statement to check if its 4 digits
        if (fourDigitCheck != 4) {
          System.out.println("NOT FOUR DIGITS");//prints that it is not
          return(accNumCheck);//returns accNumCheck
        }

        //asks user to reenter their pin, gets user input and declares new variable
        System.out.print("Please reenter your four digit pin: ");
        userPinNumCheck = sc.nextInt();

        //if statement to check if its the same pin
        if (userPinNum != userPinNumCheck) {
          System.out.println("NOT THE SAME PIN");//prints that it is not
          return(accNumCheck);//returns accNumCheck
        }

        //asks user to input first name, gets input
        System.out.print("Please enter your first name: ");
        firstName = sc.next();

        //asks user to input last name, gets input
        System.out.print("Please enter your last name: ");
        lastName = sc.next();

        //tells user they need to reenter all the information so they can log in
        System.out.print("(Press 1 then reenter credidentials to log in)\n");

        out.write("ACC NUM ~\n" + formattedAccNum);//writes account number header and account number to
file

        out.newLine();//new line

        out.write("PIN ~\n" + userPinNum);//writes pin header and pin to file

        out.newLine();//new line

        out.write(firstName + " " + lastName);//writes first name and last name

        out.close();//closes file
```

```
    }

    catch (Exception e)
    {
      System.out.println("ERROR");//error statement
    }

    return (accNumCheck);//returns to main
  }


  /*
  checkDetails(int accNumCheck, int accPinCheck)
  returns accNumCheck
  int accNumCheck - checks if the account num is valid, int accPinCheck - checks if the pin num is valid
  This method checks if someones account is valid
  */

  public static int checkDetails (int accNumCheck, int accPinCheck) {
    Scanner sc = new Scanner(System.in);

    String throwAway, pinNumCheck;//declares/initilizes variables
    int intPinNumCheck;//declares/initilizes variables

    try
    {
      BufferedReader in = new BufferedReader(new FileReader(loginAccNum));

      //reads first 3 lines as throwaways because the 4th is always going to be the pin number
      throwAway = in.readLine();
      throwAway = in.readLine();
      throwAway = in.readLine();
      pinNumCheck = in.readLine();

      intPinNumCheck = Integer.parseInt(pinNumCheck);//converts line into int

      //if statement to check if input is the same as the pin number
      if (accPinCheck == intPinNumCheck) {
        System.out.println("LOGGED IN");//outputs logged in
        validAccount = 1;
      } else {
        System.out.println("Not a valid pin number");//outputs not valid

        return(accNumCheck);//returns to main
      }

      in.close();//closes file

    }
    catch (IOException e)
    {
      System.out.println("Customer doesnt exist");//error statement
    }

    return (accNumCheck);//returns to main
  }


  /*
  checkBalance (int accNumber)
  returns accNumber
  int accNumber, imports the acc number to find the file
  This method checks individual account balances
  */

  public static int checkBalance (int accNumber) {
    Scanner sc = new Scanner(System.in);

    String accNumConvert = "", throwAway, cheqBalance, savBalance, totalBalance;//declares/initilizes
variables

    int balanceCheck = 0;//declares/initilizes variables

    accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

    //displays options and asks for users inputs
    System.out.println("Enter 1 to check chequing account balance");
    System.out.println("Enter 2 to check savings account balance");
    balanceCheck = sc.nextInt();//gets input

    try
    {
      BufferedReader in = new BufferedReader(new FileReader(accNumConvert));

      //for chequing account
      if (balanceCheck == 1) {

        //reads and outputs the chequing balance
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        cheqBalance = in.readLine();
        System.out.println(cheqBalance);

       //for savings account
      } else if (balanceCheck == 2) {
        //read and outputs savings balance
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        throwAway = in.readLine();
        savBalance = in.readLine();
        System.out.println(savBalance);

      }

    }

    catch (IOException e)
    {
      System.out.println("Account does not exist");//error message
    }


    return (accNumber);//returns to main
  }

  /*
  deposit (int accNumber)
  returns accNumber
  int accNumber, imports the acc number to find the file
  This method deposits money
  */

  public static int deposit (int accNumber) {
    Scanner sc = new Scanner(System.in);

    String accNumConvert = "", readLine = "";//declares/initilizes variables

    int accType = 0, depositAmount = 0, balance, newBalance;//declares/initilizes variables

    accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

    //header to display options
    System.out.println("Enter 1 to deposit to your chequing account");
    System.out.println("Enter 2 to deposit to your savings account");
    accType = sc.nextInt();//gets user input

    //asks how much the user would like to deposit
    System.out.println("How much money would you like to deposit into your account?");
    depositAmount = sc.nextInt();//gets number
```

```
try
{
  BufferedReader in = new BufferedReader(new FileReader(accNumConvert));
  BufferedWriter out = new BufferedWriter(new FileWriter(accNumConvert, true));//sets .txt files

  //for chequing
  if (accType == 1) {

    //finds chequing balance
    while(!readLine.equals("CHEQUING"))
    {
      readLine = in.readLine();
    }

    //if statement for if readLine is not null
    if (readLine != null) {
      //reads balance
      readLine = in.readLine();

      balance = Integer.parseInt(readLine);//declares balance while converting line into int

      newBalance = balance + depositAmount;//new balance is created by adding the deposit to balance

    }

  //for savings
  } else if (accType == 2) {

    //loop to find savings
    while(!readLine.equals("SAVINGS"))
    {
      readLine = in.readLine();//reads lines
    }

    if (readLine != null) {
      readLine = in.readLine();//reads balance

      balance = Integer.parseInt(readLine);//declares balance while converting line into int

      newBalance = balance + depositAmount;//new balance is created by adding the deposit to balance

    } else {
      System.out.println("No accounts found");//reads if there isnt an account open

    }

  }
}
```

```
  catch (Exception e)
  {
    System.out.println("Error reading file");//error message
  }
  return (accNumber);//returns to main

}


/*
withdraw (int accNumber)
returns accNumber
int accNumber, imports the acc number to find the file
This method withdraws money
*/

public static int withdraw (int accNumber) {
  Scanner sc = new Scanner(System.in);//scanner

  String accNumConvert = "", readLine = "";//declares/initilizes variables

  int accType = 0, withdrawAmount = 0, balance, newBalance;//declares/initilizes variables

  accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

  //header to display options
  System.out.println("Enter 1 to withdraw from your chequing account");
  System.out.println("Enter 2 to withdraw from your savings account");
  accType = sc.nextInt();//gets user input

  //asks how much money user would like to withdraw
  System.out.println("How much money would you like to withdraw from your account?");
  withdrawAmount = sc.nextInt();//gets number

  try
  {
    BufferedReader in = new BufferedReader(new FileReader(accNumConvert));
    BufferedWriter out = new BufferedWriter(new FileWriter(accNumConvert, true));//sets .txt files

    //for chequing
    if (accType == 1) {

      //finds chequing balance
      while(!readLine.equals("CHEQUING"))
      {
        readLine = in.readLine();//reads lines
      }

      if (readLine != null) {
            readLine = in.readLine();//reads balance

            balance = Integer.parseInt(readLine);//declares balance while converting line into int

            //if the amount of money in balance is less than the desired withdraw
            if (withdrawAmount < balance) {
              newBalance = balance - withdrawAmount;//subtracts from balance, creates new balance



            } else {
              System.out.println("Not enough money in account. Cannot withdraw.");//outputs if the ammount of
money in balance is less than desired withdraw

            }
          }

        //for savings
        } else if (accType == 2) {

          //loop to find savings balance
          while(!readLine.equals("SAVINGS"))
          {
            readLine = in.readLine();//reads lines
          }

          if (readLine != null) {
            readLine = in.readLine();//reads balance

            balance = Integer.parseInt(readLine);//declares balance while converting line into int

            //if the amount of money in balance is less than the desired withdraw
            if (withdrawAmount < balance) {
              newBalance = balance - withdrawAmount;//subtracts from balance, creates new balance



            } else {
              System.out.println("Not enough money in account. Cannot withdraw.");//outputs if the ammount of
money in balance is less than desired withdraw

            }
          }

        }
      }

      catch (Exception e)
      {
```

```
    System.out.println("Error reading file");//error message
  }
  return (accNumber);//returns to main
}

/*
closeAccount (int accNumber)
returns accNumber
int accNumber, imports the acc number to find the file
This method closes an account
*/

public static int closeAccount (int accNumber) {
  Scanner sc = new Scanner(System.in);//scanner

  String accNumConvert = "", throwAway;//declares/initilizes variables

  String lineRead[] = new String[16];//sets array

  int accType = 0, lineCount = 0;//declares/initilizes variables

  accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

  //header to display options
  System.out.println("Enter 1 to close a chequing account");
  System.out.println("Enter 2 to close a savings account");
  accType = sc.nextInt();//gets user input

  try
  {
    BufferedReader in = new BufferedReader(new FileReader(accNumConvert));
    BufferedWriter out = new BufferedWriter(new FileWriter(accNumConvert, true));//sets .txt files

    //for chequing
    if (accType == 1) {
      //loop if line isnt equal to null
      while (lineRead != null) {
        lineRead[lineCount] = in.readLine();//reads lines and sets to array

        //if statement if the line is chequing
        if (lineRead.equals("CHEQUING")) {
          lineRead[lineCount] = "";//removes writing from file
          throwAway = in.readLine();//throwaway
          throwAway = in.readLine();//throwaway

          System.out.println("Chequing Account Closed");//outputs that the account is closed

          return(accNumber);//returns to main
        }
```

```
            lineCount++;//adds 1 to line count
          }
        //for savings
        } else if (accType == 2) {
          //loop if line isnt equal to null
          while (lineRead != null) {
            lineRead[lineCount] = in.readLine();//reads lines and sets to array

            //if statement if the line is chequing
            if (lineRead.equals("SAVINGS")) {
              lineRead[lineCount] = "";//removes writing from file
              throwAway = in.readLine();//throwaway
              throwAway = in.readLine();//throwaway

              System.out.println("Savings Account Closed");//outputs that the account is closed

              return(accNumber);//returns to main
            }
            lineCount++;//add 1 to line count
          }
        }

    }

    catch (IOException e)
    {
      System.out.println("Error reading file");//error message
    }
    return (accNumber);//returns to main
  }

  /*
  openAccount (int accNumber)
  returns accNumber
  int accNumber, imports the acc number to find the file
  This method opens an account
  */

  public static int openAccount (int accNumber) {
    Scanner sc = new Scanner(System.in);//scanner

    String accNumConvert = "";//declares/initilizes variables
    int accType = 0;//declares/initilizes variables

    accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

    //header to display options
    System.out.println("Enter 1 to open a chequing account");
    System.out.println("Enter 2 to open a savings account");
    accType = sc.nextInt();//gets user input

    try
    {
      BufferedReader in = new BufferedReader(new FileReader(accNumConvert));//sets .txt file

      BufferedWriter out = new BufferedWriter(new FileWriter(accNumConvert, true));//sets .txt file

      //for chequing
      if (accType == 1) {
          //writes a chequings header and a starting balance of 0 to file
          out.write("");
          out.write("CHEQUING");
          out.write("0");
          System.out.println("Account Opened");//outputs that the account is open
      } else if (accType == 2) {
          //writes a savings header and a starting balance of 0 to file
          out.write("");
          out.write("SAVINGS");
          out.write("0");
          System.out.println("Account Opened");//outputs that the account is open
      }


    }

    catch (IOException e)
    {
      System.out.println("Error reading file");//error statement
    }


    return (accNumber);//returns to main
  }

  /*
  changePin (int accNumber)
  returns accNumber
  int accNumber, imports the acc number to find the file
  This method changes your pin
  */

  public static int changePin (int accNumber, int currentPin) {
    Scanner sc = new Scanner(System.in);//scanner

    String accNumConvert = "";//declares/initilizes variables

    String lineRead = "", throwAway, newPinString;//declares/initilizes variables
```

```
String[] reWriteFile;//declares/initilizes variables

int lineCount = 0, counter = 0, currentPinCheck = 0, newPin = 0;//declares/initilizes variables

accNumConvert = Integer.toString(accNumber);//declares/initilizes variables

//asks for the current pin
System.out.println("Enter your current pin to change your pin");
currentPinCheck = sc.nextInt();//gets user input

//if the pin matches
if (currentPinCheck == currentPin) {
  System.out.println("Enter the pin you would like to change to");//asks for pin user wants to change to
  newPin = sc.nextInt();//gets new pin

  try
  {
    BufferedReader in = new BufferedReader(new FileReader(accNumConvert));

    //reads to see if text exists in file
    while (lineRead != null) {
      lineRead = in.readLine();//reads lines
      lineCount++;//adds 1 to line count
    }

    reWriteFile = new String[lineCount];//declares/initilizes variables

    //loop for finding the pin number
    while (counter < 3) {
      reWriteFile[counter] = in.readLine();//reads number
      counter++;//adds 1 to counter
    }

    newPinString = Integer.toString(newPin);//declares new pin

    reWriteFile[counter] = newPinString;//replaces old pin to new pin
    counter++;//adds 1 to counter

    //loop for finding pin
    for (int i = counter; i < lineCount; i++) {
      reWriteFile[counter] = in.readLine();//reads line
    }


    BufferedWriter out = new BufferedWriter(new FileWriter(accNumConvert, true));//sets .txt files

    //loop for finding pin number
    for (int i = 0; i < lineCount; i++) {
      out.write(reWriteFile[counter]);//writes new pin
```

```
          }


        }

        catch (IOException e)
        {
          System.out.println("Error reading file"); //error statement
        }

      //if pin doesn't match
      } else {
        System.out.println("Not the same pin.");//outputs that the pins are not the same

      }

      return (accNumber);//returns to main
    }





    public static void main (String[]args) {
      Scanner sc = new Scanner(System.in);//scanner


      String nextAccNum;//declares/initilizes variables

      int logInChoice, accNumLog = 0, accPinLog = 0, cancelLoop = 0, accountOptionSelection = 0, logOut =
0;//declares/initilizes variables

      validAccount = 0;//declares/initilizes variables

      //infinite loop for choices
      while (cancelLoop != -1) {

        //outputs header
        System.out.println("BANKING SIMULATOR");
        System.out.println("-----------------");

        //outputs options and user intputs an option
        System.out.println("Enter 1 to log in. Enter 2 to create a new account. Enter 3 to end program.");
        logInChoice = sc.nextInt();

        //if they choose to log in
        if (logInChoice == 1) {
          System.out.print("Enter your account number: ");//asks for account number, user inputs
          accNumLog = sc.nextInt();
```

```
            loginAccNum = Integer.toString (accNumLog);

            //asks user to enter their pin
            System.out.print("Enter your pin number: ");
            accPinLog = sc.nextInt();

            checkDetails(accNumLog, accPinLog);//goes to method to see if it is a valid account

        } else if (logInChoice == 2) {
            newAccount(0, 0);//goes to method
        } else if (logInChoice == 3) {
            cancelLoop = -1;//ends program
        }

        //infinite loop for options after user logs in
        if (validAccount == 1) {
            do
            {
                //header to display their options
                System.out.println("\nEnter 1 to check account balance");
                System.out.println("Enter 2 to deposit money");
                System.out.println("Enter 3 to withdraw money");
                System.out.println("Enter 4 to close an account");
                System.out.println("Enter 5 to open an account");
                System.out.println("Enter 6 to change your pin");
                System.out.println("Enter 7 to log out");
                accountOptionSelection = sc.nextInt();

                //goes to each method depending on their choice
                if (accountOptionSelection == 1) {
                    checkBalance(accNumLog);//method checks balance

                } else if (accountOptionSelection == 2) {
                    deposit(accNumLog);//method for depositing

                } else if (accountOptionSelection == 3) {
                    withdraw(accNumLog);//method for withdrawing

                } else if (accountOptionSelection == 4) {
                    closeAccount(accNumLog);//method to close account

                } else if (accountOptionSelection == 5) {
                    openAccount(accNumLog);//method to open chequing or saving

                } else if (accountOptionSelection == 6) {
                    changePin(accNumLog, accPinLog);//method to change pin number

                } else if (accountOptionSelection == 7) {
                    logOut = -1;
                    validAccount = 0;//ends loop and goes back to login options

                  }

                }
                while (logOut != -1);

              }


            }

          }


        }
```