Foundations of Programming 1 **Final Project**

Please solve the problem(s) alone. Please write the algorithm which is a document that can be handed over to a programmer who will be get all the information to write the code. Please include this document in a text file or PDF document along with the code submitted separately. Remember to test your solution thoroughly. **This project will take a lot of time. Start working on whatever pieces you can as soon as possible.** Code that does not work correctly or does not compile will lose credit. Please submit a digital copy, by the due date and time, via Canvas. The digital copy of your code should be a zip file containing the project folder named with your last name followed by the project number (in this case final). For example, canettifinal.zip would be my zip file for the final project. Good luck!

In this project you will tie together many of the concepts that you have learned throughout the course into a unified banking application. You will develop a program that performs many common bank transactions that a bank employee or customer might perform on one or more accounts.

When executed, the program will display a menu with the following choices as follows:

```
============================================================
What do you want to do?
1. Open an account
2. Get account information and balance
3. Change PIN
4. Deposit money in account
5. Transfer money between accounts
6. Withdraw money from account
7. ATM withdrawal
8. Deposit change
9. Close an account
10. Add monthly interest to all accounts
11. End Program
============================================================
```

The program will continuously loop and prompt the user to enter the number of the transaction to perform and then perform the transaction. If the user enters '11' the program should end. If the user enters an invalid option, the program should display "Invalid choice" and let the user try again.

Each option should call a method to implement the details of that transaction. That method will need to call additional methods or classes to fully complete the transaction. Details of each option/transaction are described below.

The class that contains your main program will be called BankManager. You must utilize at least four additional classes. A class called Account, that represents a single bank account, a class called Bank, that represents the bank and stores all the accounts in the bank, a class called CoinCollector, that represents a coin collecting machine, and a class called BankUtility that will provide several helper methods to be used throughout your program. Details of these classes are described next.

BankManager class

The BankManager class is where you will implement your 'main' method and start the program from. The BankManager class should create an instance of a Bank object when the program runs and use that instance to manage the Accounts in the bank. The BankManager class must implement the following methods:

- Your 'main' method. The main method should display the menu and continually loop, performing the transactions, until the user enters 11 to exit the program. If the user enters a number from 1-10, the program should call a method that implements the functionality for that transaction. See details of each transaction at the end of this document.
- A method, 'promptForAccountNumberAndPIN' that takes one parameter, a Bank object that represents the bank. The method should prompt the user to enter an account number and then try to find a matching account with that account number in the bank. If an account cannot be found, the program should print "Account not found for account number: 12345678" (assuming the user entered 12345678) and return null. If an account is found, the program should then prompt the user to enter a PIN. If the PIN entered does not match the PIN for the account, then the program should print "Invalid PIN" and return null. If the PIN matches, then the method should return the Account object. **This method will be needed for MOST (but not all) transactions.**

CoinCollector class

This class represents a machine that can count change and has one method:

- A method called 'parseChange' that takes one parameter, a String that represents a set of coins/change. The method must look at each character in the String and calculate the amount it represents in cents as a 'long' and return it. The following characters represent valid coins:
    - 'P' represents a penny (1 cent)
    - 'N' represents a nickel (5 cents)
    - 'D' represents a dime (10 cents)
    - 'Q' represents a quarter (25 cents)
    - 'H' represents a half-dollar (50 cents)
    - 'W' represents a whole dollar (100 cents)

Account Class

The Account class must store the following attributes:

- Account number – a randomly generated 8-digit integer that cannot start with a 0
- Owner First Name – a String that contains the first name of the account owner
- Owner Last Name – a String that contains the last name of the account owner
- Social Security Number – a String (not an integer) that contains the 9 digits of the account owner's social security number (Note: do not use real social security numbers – use 999 or 000 as the first three digits for this program)
- PIN – a String that represents the account owner's 4-digit personal identification number – randomly generated upon account creation and may start with one or more zeroes
- Balance – a number that represents the account balance **in cents**.

The account class **must** implement the following methods:

- Getters and setters to get and set each attribute as described above
    - (e.g. for 'ownerFirstName', you must implement 'getOwnerFirstName' and 'setOwnerFirstName'
- A method 'deposit' that takes one parameter - an amount to deposit into the account as a 'long'. The method then adds the amount to the account balance and returns a 'long' representing the new account balance
- A method 'withdraw' that takes one parameter - an amount to withdraw from the account as a 'long'. The method then subtracts the amount from the account balance and returns a 'long' representing the new account balance
- A method 'isValidPIN' that takes one parameter – A String that represents a PIN. The method then compares the PIN that was passed in against the PIN that is on the account. If the PINs match, it returns true, otherwise, it returns false.
- A method 'toString' that has no parameters but returns a String that contains the names and values of all of the attributes as follows. This method should NOT print out anything but can be invoked as part of a System.out.println elsewhere.

You may add other methods as you wish to this class.

```
============================================================

Account Number: 79543812

Owner First Name: George

Owner Last Name: Washington

Owner SSN: XXX-XX-1776

PIN: 3759

Balance: $0.00

============================================================
```

Bank Class

The Bank class must store the following attributes

- An array (or list, dict, etc) of Accounts
    - The array represents all the accounts in the bank.
    - The bank must support up to 100 accounts when submitted
- A constant that represents the number of accounts supported by the bank (you should set this to a low value during development to help test running out of accounts)

The Bank class must implement the following methods:

- A method, 'addAccountToBank', that takes one parameter, an Account object, to add to the array of accounts in the Bank. The method should iterate through all the accounts in the array until it finds an empty/null index. It should then add the account to that index in the array to represent a new account that was opened and return true to indicate the account was successfully added. If there is no more room for the Bank to accept any more accounts (i.e. there are already 100 accounts in the array). The method should print a message saying, "No more accounts available" and should return false.
- A method, 'removeAccountFromBank', that takes one parameter, an Account object, to remove from the accounts array. The method should iterate through all the accounts in the array and try to match the account provided to the accounts in the Bank by the account number. If the account number of the provided account matches the account number of the Account in the array, then that index of the array should be marked 'null' to indicate that the account is closed and no longer exists.
- A method, 'findAccount' that takes one parameter, an int representing the account number. The method should iterate through all the accounts in the array and try to match the account number provided to the accounts in the Bank by the account number. If a match is found, the Account object should be returned. If an account with the provided account number does not exist, the method should return 'null' to indicate a matching Account was not found
- (Optional for EXTRA CREDIT): A method, 'addMonthlyInterest' that takes one parameter, a 'number' representing the percentage of the **annual** interest rate (e.g. if the rate is 2.5%, the value entered would be 2.5). The method should then iterate through all the accounts in the array and deposit a **monthly** interest payment into every account. The monthly interest for the account is calculated by taking the current balance and multiplying a **monthly** interest rate.

BankUtility Class

This class should provide several utility methods to use throughout the program:

- A method isNumeric will be provided for you. You may utilize this method to determine if a String is a number. If it is a number, it will return true, otherwise, it will return false. **You do NOT need and should not modify this method. It is free for you to use.**
- A method 'promptUserForString' that takes one parameter, a String that represents a prompt to print on the screen (e.g. "Enter your name"). The method should then read a line of input from the keyboard and return as a String.
- A method 'promptUserForPositiveNumber' that takes one parameter, a String that represents a prompt to print on the screen (e.g. "Enter a number"). The method should then read a number from the keyboard. If the number is less than or equal to 0, it should print a message and say "Amount cannot be negative. Try again" and continue to loop. If the number is positive, the method should return that number as a number.
- A method 'convertFromDollarsToCents' that takes one parameter, a number that represents an amount of money in dollars (e.g. 3.57) and converts to a long (e.g. 357) and returns it.
- A method 'generateRandomInteger' that takes two parameters, an integer representing the minimum value and an integer representing the maximum value of the range to generate a random number for and then return the number generated as an int. For example, if you invoke generateRandomInteger(0,5), the method should return a random number from 0-5 (including both 0 and 5).

You may add additional methods to this class as needed to assist or simplify your code.

Transactions

For each menu option, your program should perform a different transaction as follows:

1. Open an account

For open account, the program should prompt the user to enter their first and last name and their social security number:

```
OPEN ACCOUNT

Enter Account Owner's First Name:

George

Enter Account Owner's Last Name:

Washington

Enter Account Owner's SSN (9 digits):

999741776
```

Once the above information is entered, a new `Account` object should be created, and the six fields of the `Account` object should be populated using the setter methods based on the criteria defined above under the Account class. Any special logic (i.e. to generate a random number) should happen BEFORE calling the setter. Once the account is initialized, the `toString()` method on the account should be involved to print out the `Account` fields as follows:

```
============================================================

Account Number: 75927164

Owner First Name: George

Owner Last Name: Washington

Owner SSN: XXX-XX-1776

PIN: 3198

Balance: $0.00

============================================================
```

The program must check if an account already exists with that account number. If so, it must loop and try a different 8-digit account number until it finds a number that does not already exist as an account. It must then add this account to the array of accounts in the `Bank` so that `Bank` can keep track of it.

Then the program returns to the main menu.

2. Get account information and balance

For ‘Get account information and balance’, the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should print the account information and balance formatted as shown below with the dollar sign and cents after the decimal.

```
Enter account number:

75927164

Enter PIN:

3198

============================================================

Account Number: 75927164

Owner First Name: George

Owner Last Name: Washington

Owner SSN: XXX-XX-1776

PIN: 3198

Balance: $0.00

============================================================
```

Then the program returns to the main menu.

3. Change PIN

For ‘Change PIN’, the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should prompt the user to enter a new PIN number twice as show below. If the PIN numbers match, the PIN should be updated in the account and the “PIN updated” message should be displayed.

```
Enter account number:

95705452

Enter PIN:

3958

Enter new PIN:

1776

Enter new PIN again to confirm:

1776

PIN updated
```

Then the program returns to the main menu.

4. Deposit money into account

For ‘Deposit money into account’, the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should prompt the user to enter the dollars and cents to deposit as a decimal as shown below. If the amount is less than or equal to 0, print a message that says “Amount cannot be negative. Try again.” Otherwise, convert the amount entered to a `long` and call the `deposit` method on the `Account`. Then the program should print the updated balance of the account.

```
Enter account number:

37658351

Enter PIN:

4444

Enter amount to deposit in dollars and cents (e.g. 2.57):

5.25

New balance:$5.25
```

Then the program returns to the main menu.

5. Transfer money between accounts

For 'Transfer money between accounts', the program should prompt the user to enter their account number and then the PIN for the account number to transfer from and then the account number and PIN of the account to transfer to. See the error messages section for the messages to display if the account number or PIN are invalid. If the account numbers are valid and the PINs match the PINs on the accounts, the program should prompt the user to enter the dollars and cents to transfer as a decimal as shown below. If the amount is less than or equal to 0, print a message that says "Amount cannot be negative. Try again." Otherwise, convert the amount entered to a `long` and call the `withdraw` method on the "from" `Account` and then the `deposit` method on the "to" `Account.` Then the program should print the updated balances of both accounts.

```
Account to Transfer From:
Enter account number:
37658351
Enter PIN:
4444
Account to Transfer To:
Enter account number:
36279555
Enter PIN:
8571
Enter amount to transfer in dollars and cents (e.g. 2.57):
2.50
Transfer Complete
New balance in account:37658351 is:$2.75
New balance in account:36279555 is:$2.50
```

Then the program returns to the main menu.

6. Withdraw money from an account

For 'Withdraw money from account', the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should prompt the user to enter the dollars and cents to withdraw as a decimal as shown below. If the amount is less than or equal to 0, print a message that says "Amount cannot be negative. Try again." Otherwise, convert the amount entered to a `long` and call the `withdraw` method on the `Account`. Then the program should print the updated balance of the account.

```
Enter account number:
35816930
Enter PIN:
2010
Enter amount to Deposit in dollars and cents (e.g. 2.57):
123.45
New balance:$123.45
```

Then the program returns to the main menu.

7. Make an ATM withdrawal from an account

For ‘Make an ATM withdrawal from an account’, the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should prompt the user to enter the amount to withdraw in whole dollars as a multiple of $5 as shown below.

If the amount is less than 5, greater than 1000, or not divisible by 5, print a message that says “Invalid amount. Try again.”

Otherwise, calculate the number of $20 bills, $10 bills and $5 required to equal the amount withdrawn, print the amounts on the screen and then call the `withdraw` method on the `Account`.

Then the program should print the updated balance of the account.

```
Enter account number:

37658351

Enter PIN:

4444

Enter amount to withdraw in dollars (no cents) in multiples of
$5 (limit $1000):

135

Number of 20-dollar bills:6

Number of 10-dollar bills:1

Number of 5-dollar bills:1

New balance: $367.75
```

8. Deposit change into an account

For 'Deposit change', the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should prompt the user to enter a String representing a set of coins as shown below.

- 'P' represents a penny (1 cent)
- 'N' represents a nickel (5 cents)
- 'D' represents a dime (10 cents)
- 'Q' represents a quarter (25 cents)
- 'H' represents a half-dollar (50 cents)
- 'W' represents a whole dollar (100 cents)

If any characters are invalid coins (e.g. X), the program should print "Invalid coin: X". Otherwise, convert the String into the appropriate number of cents to a `long` and call the `deposit` method on the `Account`. Then the program should print the updated balance of the account.

```
Enter account number:

47322673

Enter PIN:

4919

Deposit coins:

QPDNNDXHW

Invalid coin: X

$2.06 in coins deposited into account

New balance: $2.06
```

In the above example, there are the following:

1 quarter (Q) + 2 dimes (D) + 2 nickels (N) + 1 half-dollar (H) + 1 dollar (W) + 1 penny (P) = 2.06

The 'X' is an invalid coin and is not counted.

Then the program returns to the main menu.

9. Close an account

For ‘Close an account’ the program should prompt the user to enter their account number and then the PIN for the account number. See the error messages section for the messages to display if the account number or PIN are invalid. If the account number is valid and the PIN matches the PIN on the account, the program should look through all of the `Account` objects in the `Bank`. If a matching `Account` is found, the `Bank` should remove that `Account` from its set of accounts by setting it to null

```
Enter account number:
47322673
Enter PIN:
4919
Account 47322673 closed
```

If the user tries to find the account afterwards, it should not be found:

```
Enter account number:
47322673
Account not found for account number: 47322673
```

10. Add monthly interest to all accounts

For ‘Add monthly interest to all accounts’, the program should prompt the user to enter an annual interest rate. The program should then calculate a **monthly** interest payment based on the annual interest rate for each account based on its current balance and then deposit that amount into that account.

In the example below, the first account had an original balance of $500, the second account had an original balance of $2500.

```
Enter annual interest rate percentage (e.g. 2.75 for 2.75%):

1.25

Deposited interest:$0.52 into account number:47534813, new
balance:$500.52

Deposited interest:$2.60 into account number:36069443, new
balance:$2,502.60
```

11. End Program

If the enters 11, the program should exit the loop and end the program.

Error Messages / Validation

The following shows common error handling scenarios that could occur in many of the above transactions

Invalid Account Number

If an invalid account number is entered:

```
Enter account number:

12345678

Account not found for account number: 12345678
```

Invalid PIN

If an invalid PIN is entered:

```
Enter account number:

10489423

Enter PIN:

5555

Invalid PIN
```

Negative Dollar Amount

If a negative dollar amount is entered (for deposit, withdrawal or transfer):

```
Enter amount to deposit in dollars and cents (e.g. 2.57):

-23.45

Amount cannot be negative.  Try again.

Enter amount to deposit in dollars and cents (e.g. 2.57):
```

PIN is not a number or not 4 digits

When changing your PIN, if the PIN is not 4 digits:

```
Enter account number:
37658351
Enter PIN:
3333
Enter new PIN:
blah
blah is not a number.
PIN must be 4 digits, try again.
Enter new PIN:
```

PINs do not match

When changing your PIN, if the PINs do not match:

```
Enter new PIN:
1234
Enter new PIN again to confirm:
5678
PINs do not match, try again.
Enter new PIN:
```

Insufficient Funds

When attempting to withdraw or transfer funds and there is insufficient money in the account:

```
Enter account number:
37658351
Enter PIN:
4444
Enter amount to withdraw in dollars and cents (e.g. 2.57):
100
Insufficient funds in account 37658351
```

SSN is not 9 digits

When attempting to enter a social security number for an account and the number is not 9 digits:

```
OPEN ACCOUNT

Enter Account Owner's First Name:

GK

Enter Account Owner's Last Name:

GgKk

Enter Account Owner's Social Security Number (9 digits):

4567

Social Security Number must be 9 digits
```

Grading

Your grade on this assignment (300 points + 30 points extra credit) will be defined as follows:

- The algorithm is written correctly enough to give all required details (20 points)
- Program properly handles opening new accounts (30 points)
- Program handles retrieving account information and balance (20 points)
- Program handles changing the PIN number for an account (20 points)
- Program handles depositing money in an account (20 points)
- Program handles transferring money between accounts (30 points)
- Program handles an withdrawing money from account (20 points)
- Program handles an ATM withdrawal (30 points)
- Program handles counting coins and depositing change to account (30 points)
- Program handles closing an account (20 points)
- Program loops correctly through main menu and ends the program (20 points)
- Program has appropriate comments, including header comments (15 points)
- Program follows course coding standard guidelines (15 points)
- Program handles depositing monthly interest to all accounts (30 points – EXTRA CREDIT)