**HW4: Bank Accounts: Using ArrayLists**

You have been hired as a programmer by a major bank. Your first project is a small banking transaction system. Each account consists of a number and a balance. The user of the program (the teller) can create a new account, as well as perform deposits, withdrawals, balance inquiries, close accounts, etc.**(Note: Some additional features have been added for this assignment.)**

For this assignment, **you must use Classes** and move functionality into the classes. Specifically, you should have at least the following classes:

1. A **Bank** class which consists of an **ArrayList of Accounts** currently active or closed.
2. An **Account** class which consists of a **Depositor, an account number, an account type, account status (open or closed), account balance, an ArrayList of TransactionReceipts performed on the account. (Note: creating an account is considered a transaction.)**
3. A **Depositor** class which has a **Name and a social security number**.
4. A **Name** class which consists of **first and last names**.
5. A **Check** class with data fields consisting of **an account number, the check amount, and a dateOfCheck**
6. A **TransactionTicket** class with data fields consisting of **an account number, a dateOfTransaction, typeOfTransaction (deposit, withdrawal, balance inquiry, new account, delete account, etc.), amountOfTransaction (for deposits and withdrawals), termOfCD (6, 12, 18, or 24 months - see below)**.
7. A **TransactionReceipt** class with data fields consisting of **a TransactionTicket, successIndicatorFlag, reasonForFailure String, accountType, preTransactionBalance, postTransactionBalance, postTransactionMaturityDate (for CDs).**

As before, you must implement appropriate methods in each class so as to implement the functionality required. New methods are to be added as necessary.

The data members of each class must be private. As such, you may need to provide accessor and mutator methods as necessary. **Any setters (mutators) used must be private.**

**Remember, all I/O should be done only in the methods of the class that contains the main() method.**

As in previous assignments, initially, the account information of existing customers is to be read into the database. **(Note: the ArrayList of Accounts will grow dynamically as each account is created.)** The program keeps tracks of the actual number of currently active accounts. **A neatly formatted table (with column headings) of the initial database of active accounts should be printed.** The column headings should include: **Last Name, First Name, SSN, Account Number, Account Type, Account Status (open or closed), Balance (with a precision of 2), Maturity Date (used for CD accounts)**.

As before, the program then allows the user to select from the following menu of transactions:

Select one of the following:

W - Withdrawal
D - Deposit
C - Clear Check
N - New account **(NOTE: the ArrayList of Accounts will grow when you create a new Account)**
B - Balance
I - Account Info (without transaction history) **(NOTE: include at least one depositor who has multiple accounts)**
H - Account Info plus Account Transaction History
S - Close Account (close (shut), but do not delete the account) **(Note: no transactions are allowed on a closed account)**
R - Reopen a closed account
X - Delete Account (close and delete the account from the database)) **(NOTE: must have zero balance to delete)**
Q - Quit

Note 1: The Clear Check transaction is only valid for checking accounts. It is like a withdrawal; except, **you must also check the date of the check.** You may only clear a check if the date on the check is no more than six months ago. No post-dated checks (checks with a future date) may be cleared. Use the **Calendar class** to implement this. In addition, a check will clear only if there is sufficient funds in the account. If the account lacks sufficient funds, the check will not clear and the account will be charged a $2.50 Service Fee for "bouncing" a check.

Note 2: **CD accounts will now contain a maturityDate.** Deposits and Withdrawals will be allowed only on or after the maturity date. When a deposit or withdrawal is made, have the user select a new maturity date for the CD. the choices are either 6, 12, 18, or 24 months from the date of the deposit or withdrawal. Again, use the **Calendar** class to implement this.

Once the user has entered a selection, appropriate methods (in the class that contains the main() method) should be called to perform the specific transaction. These methods will call the class implemented methods as necessary. **At the end, before the user quits, the program prints a neatly formatted table (as above) of the contents of the final database**.

**As in previous assignments, make sure to use enough test cases so as to completely test program functionality.**

**Make sure that there is at least one depositor that has multiple accounts at the bank.**
**Make sure that there are at least two depositors that have the same last name but different first names.**

**Minimal Class Requirements:**

1a. The Bank class should **at least** have a default (No-Arg) constructor that would allow statements of the form:

```
Bank bank = new Bank();                    //implements a No-Arg Constructor
```

1b. The Bank class should have **at least** have the following methods:

```
public TransactionReceipt openNewAcct(Account...)        //Bank creates a TransactionTicket for the new Account
public TransactionReceipt getBalance(TransactionTicket...)
public TransactionReceipt makeDeposit(TransactionTicket...)
public TransactionReceipt makeWithdrawal(TransactionTicket...)
public TransactionReceipt clearCheck(Check...)
public TransactionReceipt closeAcct(TransactionTicket...)
public TransactionReceipt reopenAcct(TransactionTicket...)
public TransactionReceipt deleteAcct(TransactionTicket...)
private int findAcct(...)                  (return value indicates index of found Account or reason for failure)
public Account getAcct(...)
public int getNumAccts(...)
```

2a. The Account class should **at least** have a constructor that would allow statements of the form:

```
Account account = new Account(...);        //implement both a No-Arg and a Parametized Constructor
```

2b. The Account class should have **at least** the following methods:

```
public TransactionReceipt getBalance(TransactionTicket...)
public TransactionReceipt makeDeposit(TransactionTicket...)
public TransactionReceipt makeWithdrawal(TransactionTicket...)
public TransactionReceipt clearCheck(Check...)
public TransactionReceipt closeAcct(TransactionTicket...)
public TransactionReceipt reopenAcct(TransactionTicket...)
public ArrayList<TransactionReceipt> getTransactionHistory(TransactionTicket...)
public Depositor getDepositor(...)
public int getAcctNumber(...)
public String getAcctType(...)
public Calendar getMaturityDate(...)
public void addTransaction(TransactionReceipt...)
```

3a. The Depositor class should **at least** have a constructor that would allow statements of the form:

```
Depositor depositor = new Depositor(...);  //implement both a No-Arg and a Parametized Constructor
```

3b. The Depositor class should **at least** the following methods:

```
public Name getName(...)
public String getSSN(...)
```

4a. The Name class should **at least** have a constructor that would allow statements of the form:

```
Name name = new Name(...);                 //implement both a No-Arg and a Parametized Constructor
```

4b. The Name class should **at least** the following methods:

```
public String getFirstName(...)
public String getLastName(...)
```

5a. The Check class should **at least** have a constructor that would allow statements of the form:

```
Check check = new Check(...);
```

6a. The TransactionTicket class should **at least** have a constructor that would allow statements of the form:

```
TransactionTicket transactionTicket = new TransactionTicket(...);
```

6b. The TransactionTicket class should **at least** the following methods:

```
public Calendar getDateOfTransaction(...)
public String getTransactionType(...)
public double getTransactionAmount(...)
Public int gettermOfCD(...)
```

7a. The TransactionReceipt class should **at least** have a constructor that would allow statements of the form:

```
TransactionReceipt transactionReceipt = new TransactionReceipt(...);
```

7b. The TransactionReceipt class should **at least** the following methods:

```
public TransactionTicket getTransactionTicket(...)
public boolean getTransactionSuccessIndicatorFlag(...)
public String getTransactionFailureReason(...)
public double getPreTransactionBalance(...)
public double getPostTransactionBalance(...)
public Calendar getPostTransactionMaturityDate(...)
```

8. Add additional constructors and methods to each class as necessary.

9. The class containing the main() method should have at least the following methods:

```
public static void main(String[] args)
public static void readAccts(...)
public static void printAccts(...);
public static void menu(...)
public static void balance(...);
public static void deposit(...);
public static void withdrawal(...);
public static void clearCheck(...);
public static void acctInfo(...);
public static void acctInfoHistory(...);
public static void newAcct(...);
public static void closeAcct(...);
public static void reopenAcct(...);
public static void deleteAcct(...);
```

The transaction methods in main() should "fill out" a TransactionTicket object, and then call the appropriate method within the Bank or Account class to carry out the requested transaction. The method should then print an appropriate transaction receipt

**10. All I/O should be done only in the methods of the class that contains the main() method.**

**Submission Requirements:**
**Create a folder on Google Drive that will contain the following:**
**1. The source files (i.e., *.java files) for each of the implemented Classes:**

```
pgmHW4.java
Bank.java; Account.java; Depositor.java, Name.java
Check.java; TransactionTicket.java; TransactionReceipt.java;
```

**2. The text file containing the initial database of accounts (e.g., initAccounts.txt)**
**3. The test cases text file (e.g., myTestCases.txt)**
**4. The output text file which contains all of the required program output (e.g., pgmOutput.txt)**
**Then, make the folder shareable and send me a link to the folder.**

**Sample Transaction Output:**

**Transaction Requested: Balance Inquiry**
**Account Number: 123456**
**Account Type: Savings Account**
**Current Balance: $200.55**

**Transaction Requested: Balance Inquiry**
**Error: Account number 999999 does not exist**

**Transaction Requested: Deposit**
**Account Number: 123456**
**Account Type: Savings Account**
**Old Balance: $200.55**
**Amount to Deposit: $100.25**
**New Balance: $300.80**

**Transaction Requested: Withdrawal**
**Account Number: 987654**
**Account Type: Checking Account**
**Current Balance: $2.33**
**Amount to Withdraw: $100.50**
**Error: Insufficient Funds Available**

**Transaction Requested: Account Info**
**SSN: 123445678**

| Last Name | First Name | SSN | Acct Num | Acct Type | Status | Balance | Maturity Date |
|---|---|---|---|---|---|---|---|
| Doe | John | 123445678 | 123456 | Savings | Open | $ 300.80 | |
| Doe | John | 123445678 | 222222 | CD | Open | $5200.55 | 05/25/2023 |
| Doe | John | 123445678 | 333333 | Checking | Open | $ 899.74 | |

**3 accounts were found**

**Transaction Requested: Clear Check**
**Account Number: 567890**
**Account Type: Checking Account**
**Old Balance: $1234.56**
**Amount of Check: $10.50**
**New Balance: $1224.06**

**Transaction Requested: Clear Check**
**Account Number: 567890**
**Account Type: Checking Account**
**Old Balance: $1224.06**
**Amount of Check: $7000.00**
**New Balance: $1221.56**
**Error: Insufficient Funds Available - Bounce Fee ($2.50) Charged**

**Transaction Requested: Clear Check**
**Account Number: 567890**
**Account Type: Checking Account**
**Old Balance: $1221.56**
**Amount of Check: $22.22**
**New Balance: $1221.56**
**Error: Check not cleared - Post-dated check: 11/22/2023**

**Transaction Requested: Account Info With Transaction History**
**SSN: 123445678**

| Last Name | First Name | SSN | Acct Num | Acct Type | Status | Balance | Maturity Date |
|---|---|---|---|---|---|---|---|
| Doe | John | 123445678 | 123456 | Savings | Open | $ 300.80 | |

***** Account Transactions *****

| Date | Transaction | Amount | Status | Balance | Reason For Failure |
|---|---|---|---|---|---|
| 11/27/2022 | Open New Account | $ 200.55 | Done | $ 200.55 | |
| 11/27/2022 | Balance Inquiry | $ 0.00 | Done | $ 200.55 | |
| 11/27/2022 | Deposit | $ 100.25 | Done | $ 300.80 | |
| 11/27/2022 | Deposit | $ -75.75 | Failed | $ 300.80 | $-75.75 is an invalid amount |
| 11/27/2022 | Clear Check | $ 10.10 | Failed | $ 300.80 | Account 123456 is not a Checking account |

| Last Name | First Name | SSN | Acct Num | Acct Type | Status | Balance | Maturity Date |
|---|---|---|---|---|---|---|---|
| Doe | John | 123445678 | 222222 | CD | Open | $5200.55 | 05/25/2023 |

***** Account Transactions *****

| Date | Transaction | Amount | Status | Balance | Reason For Failure |
|---|---|---|---|---|---|
| 11/27/2022 | Open New Account | $5000.00 | Done | $5000.00 | |
| 11/27/2022 | Balance Inquiry | $ 0.00 | Done | $5000.00 | |
| 11/27/2022 | Deposit | $ 200.55 | Done | $5200.55 | |
| 11/27/2022 | Deposit | $ 100.96 | Failed | $5200.55 | CD maturity date 05/27/2023 not reached |

| Last Name | First Name | SSN | Acct Num | Acct Type | Status | Balance | Maturity Date |
|---|---|---|---|---|---|---|---|
| Doe | John | 123445678 | 333333 | Checking | Open | $ 899.74 | |

***** Account Transactions *****

| Date | Transaction | Amount | Status | Balance | Reason For Failure |
|---|---|---|---|---|---|
| 11/27/2022 | Open New Account | $ 999.99 | Done | $ 999.99 | |
| 11/27/2022 | Withdrawal | $ 100.25 | Done | $ 899.74 | |
| 11/27/2022 | Clear Check | $ 123.45 | Done | $ 776.29 | |

**3 accounts were found**