HW3: Bank Accounts: Using Classes with Constructors and Methods

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.

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 with data fields consisting of an **array of Accounts** and the **number of accounts** currently active in the bank.
2. An **Account** class with data fields consisting of a **Depositor, an account number, an account type, a balance, and (for CD accounts) a maturityDate**.
3. A **Depositor** class with data fields consisting of a **Name and a social security number**.
4. A **Name** class with data fields consisting 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).**

You must **add appropriate methods** to each class so as to implement the functionality of each of the methods of the previous assignment (HW2). Each of the methods of HW2 should be re-implemented utilizing a class method. (You will have to decide as to which class each method belongs.) In addition, **each class should minimally have a default constructor and possibly additional parametized constructors**. (Some of the HW2 methods may become constructors.) **Any setters used should be private.**

The **data members of each class must be private**. As such, you may need to provide accessor and mutator methods.

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. The bank can handle a maximum of **MAX_NUM** accounts. 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, 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 expanded menu of transactions:

Select one of the following:

W - Withdrawal
D - Deposit
C - Clear Check
N - New account
B - Balance
I - Account Info
X - Delete Account
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.

Note 3. Use the **Calendar class** to assist you in implementing the "date" requirements.

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 constructor that would allow statements of the form:

```
Bank bank = new Bank();
```

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 deleteAcct(TransactionTicket...)
private int findAcct(...)                    (return value indicates index of found Account or reason for failure)
public Account getAcct(...)
```

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

```
Account myAccount = new Account(...);          //or account[i] = new Account()
```

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 Depositor getDepositor(...)
public int getAcctNumber(...)
public String getAcctType(...)
public Calendar getMaturityDate(...)
```

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

```
Depositor depositor = new Depositor(...);
```

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(...);
```

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 int 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 newAcct(...);
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 (i.e., the client program).

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:
pgmHW3.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	Balance	Maturity Date
Doe	John	123445678	123456	Savings	$ 300.80	
Doe	John	123445678	222222	CD	$5200.55	05/25/2024
Doe	John	123445678	333333	Checking	$ 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: 11/22/2025

Transaction Requested: Open New Account
Account number 678765 opened
Account Type: CD Account
Opening Balance: $1234.56
CD New Maturity Date: 08/03/2023

Transaction Requested: Deposit
Account Number: 666666
Account Type: CD Account
Old Balance: $1234.56
Amount to Deposit: $123.45
New Balance: $1358.01
CD New Maturity Date: 11/25/2025

Transaction Requested: Withdrawal
Account Number: 234567
Account Type: CD Account
Current Balance: $350.00
Amount to Withdraw: $100.00
Error: CD maturity date 05/25/2025 not reached

Sample Initial Accounts File Contents:

```
John Doe 123445678 123456 Savings 200.55
Jim Beam 234556789 567890 Checking 1234.56
Jane Eyre 345667890 987654 Savings 2.33
Tom Sawyer 456778901 234567 CD 500.00 7/22/2024
Huck Finn 567889012 345678 Checking 123.98
John Doe 123445678 222222 CD 5000.00 12/12/2024
John Doe 123445678 333333 Checking 999.99
Huck Finn 567889012 654321 Savings 543.66
Jack Spratt 678990123 785609 Savings 333.33
Jane Doe 456789012 389765 Checking 888.56
Jane Doe 456789012 123123 Savings 8765.43
```