HW2: Bank Accounts - Using Classes:

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..

Initially, the account information of existing customers is to be read into an **array of BankAccount objects**. **The private data members of the BankAccount Class will include: first name, last name, social security number, account number, account type (Checking, Savings, or CD), and account balance**. The bank can handle up to **MAX_NUM** accounts. Use the following function to read in the data values:

```
public static int readAccts(BankAccount[] account);
```

This method fills up the account array by reading from an input file until EOF is reached, and returns the actual number of accounts read in (referred to as **numAccts**).

After initialization, **print the initial database of accounts**. Use method printAccts() described below.

The program then allows the user to select from the following menu of transactions:

Select one of the following:
- W - Withdrawal
- D - Deposit
- N - New account
- B - Balance
- I - Account Info
- X - Delete Account
- Q - Quit

Use the following method to produce the menu:

```
public static void menu()
```

This method only displays the menu. The **main program** then prompts the user for a selection. You should verify that the user has typed in a valid selection (otherwise print out an error message and repeat the prompt).

Once the user has entered a selection, one of the following methods should be called to perform the specific transaction. **At the end, before the user quits, the program prints the contents of the database**.

public static int findAcct(BankAccount[] account, int numAccts, int reqAccount);

This method **returns the index of reqAccount in the BankAccount array** if the account exists, and -1 if it doesn't. It is called by all the remaining methods.

```
public static void withdrawal(BankAccount[] account, int num_accts);
```

This method prompts the user for the account number. If the account does not exist, it prints an error message. Otherwise, it asks the user for the amount of the withdrawal. If the account does not contain sufficient funds, it prints an error message and does not perform the transaction.

```
public static void deposit(BankAccount[] account, int num_accts);
```

This method prompts the user for the account number. If the account does not exist, it prints an error message. Otherwise, it asks the user for the amount of the deposit.

```
public static int newAcct(BankAccount[] account, int num_accts);
```

This method prompts the user for a new account number. If the account already exists, it prints an error message. Otherwise, it adds the account to the database. **The method then prompts the user to enter the new depositor's first name, last name, social security number, the account type (Checking, Savings, or CD), and the initial opening deposit.**. The method returns the new number of accounts in the database.

```
public static int deleteAcct(BankAccount[] account, int num_accts);
```

This method prompts the user for an account number. If the account does not exist, or if the account exists but has a non-zero balance, it prints an error message. Otherwise, it closes and deletes the account. It returns the new number of accounts.

```
public static void balance(BankAccount[] account, int num_accts);
```

This method prompts the user for an account number. If the account does not exist, it prints an error message. Otherwise, it prints the account balance.

```
public static void accountInfo(BankAccount[] account, int num_accts);
```

This method prompts the user for a **social security number (SSN)**. If no account exists for this SSN, it prints an error message. Otherwise, it prints the complete account information **for all of the accounts with this SSN**. The method prints the complete account information as a **neatly formatted table** (with column headings) The column headings should include: **Last Name, First Name, SSN, Account Number, Account Type, Balance (with a precision of 2)**.

```
public static void printAccts(BankAccount[] account, int num_accts);
```

This method prints a **neatly formatted table** (with column headings) of the complete account information for every active account. The column headings should include: **Last Name, First Name, SSN, Account Number, Account Type, Balance (with a precision of 2)**.

Make sure to use enough test cases so as to completely test program functionality (see HW1).
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.

EXTRA CREDIT #1: Use nested classes:
1. A BankAccount consists of a Depositor, an account number, an account type, and a balance.
2. A Depositor has a Name and a social security number.
3. A Name consists of first and last names.

EXTRA CREDIT #2:
Use a constructor to initialize the data members of a new account (including the initial accounts of the database).
Hint: a constructor is a method that can be called.

Notes:
1. All output must be file directed
2. Only output must go to the file - not interactive prompts and menus.
3. No global variables are allowed
4. The program and all methods must be properly commented.
5. All data members of classes are to be private
6. Add accessor (getter) methods and mutator (setter) methods to all classes as appropriate
7. All I/O should be done outside of the BankAccount class implementation.
8. All I/O should be done within 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 Java program source file (i.e., *.java files) (e.g., pgmHW2.java)
2. The BankAccount Class file (e.g., BankAccount.java)
3. The text file containing the initial database of accounts (e.g., initAccounts.txt)
4. The test cases text file (e.g., myTestCases.txt)
5. The output text file which contains all of the required program output (e.g., pgmOutput.txt)
6. For EC1 - the additional implemented classes: Depositor.java, Name.java
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
  LastName  FirstName        SSN   Acct Num   Acct Type    Balance
       Doe       John  123445678     123456     Savings   $ 300.80
       Doe       John  123445678     222222          CD   $5200.55
       Doe       John  123445678     333333    Checking   $ 899.74
 3 accounts were found
```

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
Huck Finn 567889012 345678 Checking 123.98
John Doe 123445678 222222 CD 5000.00
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
```