APL Experiment 1 (C) :

Class, Object and Constructor

1C]

Working with static fields and static methods

Problem Statement:

Design a Java program that simulates a **Bank Account System** where you can create multiple bank account objects and perform operations such as deposit, withdraw, and check the balance. The program should demonstrate the usage of **static fields** and **static methods** to manage the bank's overall data (like the total number of accounts and the total balance across all accounts).

Requirements:

1. **Class Definitions:**
   - **BankAccount**: The class should represent a bank account with the following attributes:
     - Account Number (String)
     - Account Holder Name (String)
     - Balance (double)
   - **Bank**: The class should represent a bank and include the following static fields:
     - **Total Accounts** (static int): This field should keep track of the total number of bank accounts created.
     - **Total Balance** (static double): This field should track the sum of all account balances.
2. **Static Fields:**
   - In the **BankAccount** class, define static fields to track the **total number of accounts** created and the **total balance** of all accounts.
   - Each time a new account is created, update these static fields to reflect the changes:
     - Increment the total number of accounts.
     - Add the initial balance of the new account to the total balance.

3. **Static Methods:**
   - Implement a **static method** in the **BankAccount** class to display the **total number of accounts** and the **total balance** across all accounts.
   - The **BankAccount** class should also have instance methods to perform the following operations:
     - **Deposit**: Deposit money into the account.
     - **Withdraw**: Withdraw money from the account.
     - **Check Balance**: Display the current balance of the account.
4. **Operations and Flow:**
   - When a new account is created, the program should ask for the account holder's name, account number, and initial deposit amount.
   - After creating an account, allow the user to:
     - Deposit money into the account.
     - Withdraw money from the account.
     - Check the balance of the account.
   - Use static methods to show the total number of accounts and the total balance whenever requested.
5. **Sample Output:**

```
Welcome to the Bank Account System!


Choose an option:

1. Create a new account

2. Deposit money into an account

3. Withdraw money from an account

4. Check balance of an account

5. Show total accounts and total balance

6. Exit


Enter your choice (1-6): 1


Enter Account Holder's Name: John Doe

Enter Account Number: 123456
Enter Initial Deposit: 1000

Account created successfully!

Choose an option:
1. Create a new account
2. Deposit money into an account
3. Withdraw money from an account
4. Check balance of an account
5. Show total accounts and total balance
6. Exit

Enter your choice (1-6): 2

Enter Account Number: 123456
Enter Amount to Deposit: 500

Deposit successful!

Choose an option:
1. Create a new account
2. Deposit money into an account
3. Withdraw money from an account
4. Check balance of an account
5. Show total accounts and total balance
6. Exit

Enter your choice (1-6): 5

Total Accounts: 1
Total Balance: 1500.00
```

```
Choose an option:

1. Create a new account

2. Deposit money into an account

3. Withdraw money from an account

4. Check balance of an account

5. Show total accounts and total balance

6. Exit


Enter your choice (1-6): 6

Exiting the system. Thank you!
```

Guidelines:

1. **Static Fields and Methods:**
   - The totalAccounts and totalBalance fields should be updated whenever a new account is created or when any deposit/withdrawal occurs.
   - The static method should be able to access these static fields without needing an instance of the BankAccount class.
2. **Bank Account Operations:**
   - For each account, allow operations like deposit and withdrawal. Ensure that withdrawals cannot exceed the current balance.
3. **User Input and Flow Control:**
   - Use a loop to continuously prompt the user for their choice and allow them to perform operations until they choose to exit.
   - Implement input validation to handle incorrect choices and ensure the input is of the correct data type.
4. **Object-Oriented Design:**
   - Use constructors to initialize the instance variables for each BankAccount object.
   - Maintain proper encapsulation by keeping account details private and providing public methods to interact with the data.
5. **Edge Cases:**
   - Handle attempts to withdraw more money than available in the account.
   - Ensure proper formatting of the balance and display of total accounts and total balance.