**Banking System Assignment: Java Exceptions, OOP, and Collections**

**Objective:**

Design and implement a simple banking system using **Java** that incorporates the following:

* **OOP principles** such as **Encapsulation**, **Inheritance**, **Polymorphism**, and **Abstraction**.
* **Exception Handling** to deal with errors such as insufficient balance, invalid input, and account-related issues.
* **Collections** like **ArrayList** and **HashMap** to store and manage multiple bank accounts and transactions.

**Assignment Requirements:**

**1. BankAccount Class (Base Class):**

* Create a class called BankAccount that will represent a general bank account.
* Include the following attributes:
  + accountNumber (String)
  + accountHolderName (String)
  + balance (double)
* Provide the following methods in BankAccount:
  + A constructor to initialize the account.
  + deposit(double amount) – Deposit money into the account.
  + withdraw(double amount) – Withdraw money from the account, ensuring the balance is sufficient.
  + displayBalance() – Display the current balance of the account.

**2. SavingsAccount Class (Subclass of BankAccount):**

* Create a subclass of BankAccount called SavingsAccount.
* Add the following attributes:
  + interestRate (double) – Annual interest rate (e.g., 5%).
* Override the withdraw method in SavingsAccount to charge a penalty fee for withdrawals below a certain threshold (e.g., a $5 fee for withdrawals under $50).
* Add a method calculateInterest() to calculate the interest based on the current balance and interest rate.
* Demonstrate **Polymorphism** by overriding the withdraw method.

**3. TransactionException Class (Custom Exception):**

* Create a custom exception class TransactionException that will be thrown when:
  + A withdrawal attempt exceeds the current balance.
  + The deposit or withdrawal amount is invalid (negative or zero).

**4. AccountNotFoundException Class (Custom Exception):**

* Create a custom exception class AccountNotFoundException that will be thrown when an account is not found in the banking system.

**5. Bank Class:**

* Create a Bank class to manage multiple bank accounts.
  + Use a **HashMap** to store accounts with the **account number** as the key and **BankAccount** (or its subclass) as the value.
  + Provide the following methods:
    - addAccount(BankAccount account) – Add an account to the bank.
    - removeAccount(String accountNumber) – Remove an account from the bank.
    - findAccount(String accountNumber) – Find an account by its number.
    - transferFunds(String fromAccountNumber, String toAccountNumber, double amount) – Transfer money between two accounts, ensuring sufficient balance in the source account.

**6. Main Class (Menu-Driven Program):**

* Create a **menu-driven program** that allows users to:
  + Create a new account (for both BankAccount and SavingsAccount).
  + Deposit money into an account.
  + Withdraw money from an account.
  + Transfer money between accounts.
  + Check the balance of an account.
  + Calculate interest for savings accounts.
* Ensure the program uses **exception handling** to manage errors such as insufficient balance, invalid transactions, and account not found.

**7. Additional Functionality (Optional but encouraged):**

* Implement **transaction history** for each account. Store each transaction (deposit, withdrawal, and transfer) and allow users to view the transaction history.

**Expected Deliverables:**

1. **Code Implementation**:
   * Implement the classes as described above.
   * Ensure that the system is modular, uses proper OOP principles, and handles exceptions appropriately.
2. **Test Cases**:
   * Test the system by:
     + Creating bank accounts.
     + Performing deposit, withdrawal, and transfer operations.
     + Demonstrating exception handling (e.g., insufficient balance, invalid transactions, account not found).
     + Displaying balance and calculating interest.
3. **Documentation**:
   * Provide comments and documentation for each class and method.
   * Describe how exception handling works in the program and what errors are handled.

**Evaluation Criteria:**

1. **Correctness**:
   * Does the system correctly handle account creation, deposits, withdrawals, and transfers?
   * Are exceptions (e.g., insufficient funds, invalid input, account not found) handled gracefully?
2. **Object-Oriented Design**:
   * Does the program use **Encapsulation**, **Inheritance**, **Polymorphism**, and **Abstraction** effectively?
   * Are the classes designed in a modular and extensible way?
3. **Exception Handling**:
   * Are custom exceptions (TransactionException, AccountNotFoundException) used properly to handle errors?
   * Does the system handle user input errors and runtime exceptions without crashing?
4. **Code Quality**:
   * Is the code clean, readable, and well-structured?
   * Is there proper use of **Collections** (e.g., HashMap for storing accounts)?
   * Is the program robust and flexible?
5. **User Interaction**:
   * Is the menu-driven interface user-friendly and intuitive?
   * Does it provide clear error messages when necessary?

**Sample Menu for User Interaction:**

markdown

Copy

--- Banking System Menu ---

1. Create a New Bank Account

2. Create a New Savings Account

3. Deposit Money

4. Withdraw Money

5. Transfer Funds

6. Display Account Balance

7. Calculate Interest (Savings Account)

8. Exit

Enter your choice:

**Sample Expected Output:**

markdown

Copy

--- Banking System Menu ---

1. Create a New Bank Account

2. Create a New Savings Account

3. Deposit Money

4. Withdraw Money

5. Transfer Funds

6. Display Account Balance

7. Calculate Interest (Savings Account)

8. Exit

Enter your choice: 1

Enter account number: 101

Enter account holder name: Alice

Enter initial balance: 5000

--- Banking System Menu ---

1. Create a New Bank Account

2. Create a New Savings Account

3. Deposit Money

4. Withdraw Money

5. Transfer Funds

6. Display Account Balance

7. Calculate Interest (Savings Account)

8. Exit

Enter your choice: 3

Enter account number: 101

Enter deposit amount: 1500

Deposited: 1500.0

--- Banking System Menu ---

1. Create a New Bank Account

2. Create a New Savings Account

3. Deposit Money

4. Withdraw Money

5. Transfer Funds

6. Display Account Balance

7. Calculate Interest (Savings Account)

8. Exit

Enter your choice: 4

Enter account number: 101

Enter withdrawal amount: 2000

Withdrawn: 2000.0

Account Balance: 4500.0

--- Banking System Menu ---

1. Create a New Bank Account

2. Create a New Savings Account

3. Deposit Money

4. Withdraw Money

5. Transfer Funds

6. Display Account Balance

7. Calculate Interest (Savings Account)

8. Exit

Enter your choice: 8

Exiting the system.