**Banking System Assignment: OOP and Exception Handling**

To design and implement a banking system in Java that incorporates Object-Oriented Programming (OOP) principles, such as **Encapsulation**, **Inheritance**, **Polymorphism**, and **Abstraction**, while also utilizing **Exception Handling** to ensure the system handles runtime errors like invalid inputs, insufficient balance, and other potential exceptions.

**Assignment Requirements:**

You are tasked with designing a banking system with the following functionalities and constraints:

1. **BankAccount Class**:
   * A class representing a general bank account with the following attributes:
     + accountNumber (String)
     + accountHolderName (String)
     + balance (double)
   * Provide the following methods:
     + A constructor to initialize the attributes.
     + deposit(double amount) – Deposits a certain amount into the account.
     + withdraw(double amount) – Withdraws a certain amount from the account, ensuring that the balance does not become negative.
     + displayBalance() – Displays the current balance.
   * Use **encapsulation** by making the fields private and providing **getter** and **setter** methods.
2. **SavingsAccount Class** (Subclass of BankAccount):
   * A class representing a savings account with the following additional functionality:
     + interestRate (double) – the annual interest rate.
     + calculateInterest() – Calculates and displays interest based on the current balance.
   * Demonstrate **inheritance** by extending the BankAccount class.
   * Implement **polymorphism** by overriding the withdraw() method to include a condition where withdrawals below a certain amount incur a fee (e.g., if the withdrawal is less than $50, there is a $5 fee).
3. **TransactionException Class**:
   * A custom exception class called TransactionException should be created.
   * The exception will be thrown when a withdrawal exceeds the current balance or when an invalid deposit/withdrawal amount (i.e., less than or equal to zero) is entered.
4. **Bank Class**:
   * A class to represent the bank, which manages multiple accounts.
   * It should maintain a list or a map of accounts, where each account is identified by its accountNumber.
   * Provide methods to add an account, remove an account, and find an account by its number.
   * Implement a method to transfer money between two accounts and ensure that the transfer only happens if both accounts exist and there is sufficient balance.
5. **Exception Handling**:
   * Use **exception handling** to manage scenarios such as:
     + **Insufficient Balance**: When a withdrawal exceeds the current balance.
     + **Invalid Transaction**: When a user tries to deposit or withdraw a non-positive amount.
     + **Account Not Found**: When an account is not found in the bank.
   * Handle these exceptions by throwing custom exceptions (TransactionException) and displaying appropriate error messages.
6. **Main Program**:
   * In the main class, create a **Menu-Driven Program** to interact with the user.
   * Allow the user to:
     + Create a new account (for both BankAccount and SavingsAccount).
     + Deposit money into an account.
     + Withdraw money from an account.
     + Check the balance of an account.
     + Calculate interest for a savings account.
     + Transfer money between two accounts.
   * Use a **try-catch block** to handle exceptions gracefully.

**Bonus (Optional):**

* **Transaction History**: Track all transactions (deposits, withdrawals, transfers) for each account and display a transaction history when requested.

**Sample Program Structure:**

1. **BankAccount.java**
2. **SavingsAccount.java**
3. **TransactionException.java**
4. **Bank.java**
5. **Main.java**

**Expected Deliverables:**

1. **Code Implementation**:
   * Implement the classes as described above, ensuring that the system is modular, uses the OOP principles effectively, and handles exceptions properly.
2. **Test Cases**:
   * Demonstrate the functionality of your banking system by providing test cases to show:
     + Creating accounts.
     + Performing deposits and withdrawals (including invalid transactions).
     + Showing balance.
     + Calculating interest for savings accounts.
     + Handling errors like insufficient funds, invalid amounts, and non-existing accounts.
3. **Documentation**:
   * Provide brief documentation for each class and method.
   * Include comments in the code explaining important sections and logic.

**Evaluation Criteria:**

1. **Correctness**: Does the program function as expected? Are all requirements implemented correctly?
2. **Exception Handling**: Are the exceptions handled properly, with clear and useful messages?
3. **OOP Principles**: Is the program well-structured using OOP concepts like Encapsulation, Inheritance, Polymorphism, and Abstraction?
4. **Code Quality**: Is the code clean, readable, and well-organized? Is it modular and reusable?
5. **User Interaction**: Is the menu-driven interface user-friendly? Does it handle invalid user input properly?