LOW LEVEL DESIGN

ATM Interface in Java

(Console Based Application)

1. ATMInterface.java:

I. Variables:

i. scanner: Scanner object for user input.
ii. b(Object reference for Bank): Bank object to manage accounts.
iii. currentAccountHolder: AccountHolder object representing the currently logged-in user.
iv. flag: Boolean flag to control the main loop.

II. Methods:

i. main(String[] args): Entry point of the program, handles user registration, login, and feature selection.
ii. bankAccounts(): Initializes the bank with default account holders.
iii. register(): Allows users to register a new account.
iv. login(): Handles user login and authentication.
v. features(): Displays menu options and invokes corresponding operations based on user choice.
vi. displayTransactionHistory(): Displays the transaction history of the current account holder.
vii. withdraw(): Allows the user to withdraw money from their account.
viii. depositMoney(): Allows the user to deposit money into their account.
ix. transferFunds(): Facilitates fund transfer between accounts.

2. Bank.java:

Variables:

i. accountHolders: List to store AccountHolder objects.

Methods:

ii. addAccountHolder(AccountHolder account): Adds a new account holder to the bank.

iii. validateAccountHolder(String userId, String userPin): Validates user credentials during login.
iv. transferFunds(String senderId, String recipientId, double amount): Transfers funds between accounts.
v. findAccountHolder(String userId): Finds an account holder by User ID.

3. AccountHolder.java:

Variables:

i. userId: User ID of the account holder.
ii. userPin: PIN of the account holder.
iii. balance: Current balance of the account holder.
iv. transactionHistory: List to store Transaction objects.

Methods:

i. AccountHolder(String userId, String userPin): Constructor to initialize an account holder.
ii. getUserId(): Getter for User ID.
iii. getUserPin(): Getter for User PIN.
iv. getTransactionHistory(): Getter for transaction history.
v. withdraw(double amount): Allows the account holder to withdraw money.
vi. depositMoney(double amount): Allows the account holder to deposit money.
vii. addTransaction(Transaction transaction): Adds a transaction to the transaction history.

4. Transaction.java:

Variables:

i. amount: Amount of the transaction.
ii. description: Description of the transaction.

Methods:

i. Transaction(double amount, String description): Constructor to create a new transaction.
ii. toString(): Overrides the toString method to provide a string representation of the transaction.

5. Flow of Operations:

i. User interacts with ATMInterface.
ii. ATMInterface interacts with Bank and AccountHolder objects based on user input.
iii. Bank manages account holders and facilitates fund transfers.
iv. AccountHolder handles individual account operations such as withdrawals, deposits, and transaction history.
v. Transactions are recorded and managed through Transaction objects.

6. Error Handling:

i. Invalid input handling: Scanner ensures correct input types (int, double, string).
ii. Authentication: Bank validates user credentials during login.
iii. Insufficient balance checks: AccountHolder checks for sufficient balance before withdrawals.

7. Security Measures:

i. User credentials (User ID, PIN) are stored securely and validated during login.
ii. Transactions are logged in the transaction history for audit trails.
iii. Input validation prevents malicious input.

8. Extensibility:

i. Additional features can be added to AccountHolder for account management.
ii. Bank operations can be expanded for more complex banking functionalities.
iii. Transaction categorization and analytics can be implemented for reporting purposes.

9. Conclusion:

The ATM system is structured with clear separation of concerns between ATMInterface, Bank, AccountHolder, and Transaction classes. Modular design allows for easy maintenance, scalability, and future feature enhancements. Error handling and security measures ensure a robust and secure banking experience for users.