**Low-Level Design (LLD) for ATM Application**

**1. Overview**

The ATM application is designed as a console-based Java application that simulates basic ATM functionalities like login, account creation, deposits, withdrawals, transfers, and viewing transaction history. The system is structured in a modular way, using various classes to represent different aspects of the ATM and banking system, such as ATM, Bank, Account, AccountHolder, and BankTransaction.

**2. Class Diagram**

The following describes the key classes and their relationships:

1. **ATM**: Main class responsible for interacting with the user and providing ATM functionalities.
2. **Bank**: Manages the bank's account holders and verifies user authentication.
3. **AccountHolder**: Represents the account holder with user details and associated account information.
4. **Account**: Handles transactions such as deposits, withdrawals, and transfers.
5. **BankTransaction**: Represents individual transactions with details like type, amount, and date.

**3. Class Descriptions**

**3.1 Main Class**

* **Responsibilities**: The Main class initializes the Bank and ATM objects and starts the application.
* **Methods**:
  + main(String[] args): Initializes Bank and ATM instances and invokes the start() method on the ATM.

**3.2 ATM Class**

* **Responsibilities**: This class manages the core ATM operations, including login, account creation, transaction management, and menu navigation.
* **Attributes**:
  + Bank bank: A reference to the bank object to manage account holders and authentication.
* **Methods**:
  + start(): Provides the main user interface loop, displaying options like login, account creation, and exit.
  + login(): Authenticates the user based on user ID and PIN.
  + createAccount(): Allows new users to create an account by providing user information and initial deposit.
  + showMenu(AccountHolder): Displays the user’s account menu, providing options for transactions and balance inquiries.
  + withdraw(Account), deposit(Account), transfer(Account): Handle various transactions.
  + showTransactionHistory(Account): Displays the user's transaction history.

**3.3 Bank Class**

* **Responsibilities**: Manages the account holders and handles authentication.
* **Attributes**:
  + Map<String, AccountHolder> accountHolders: A map storing user details indexed by user ID.
* **Methods**:
  + addAccountHolder(AccountHolder): Adds new account holders to the bank.
  + findAccountHolder(String userId): Returns the account holder object corresponding to the given user ID.
  + authenticateUser(String userId, String userPin): Validates the user's login credentials.

**3.4 AccountHolder Class**

* **Responsibilities**: Represents an individual user of the bank with details like user ID, PIN, name, and associated account.
* **Attributes**:
  + String userId: The unique identifier for the user.
  + String userPin: The PIN associated with the user’s account.
  + String name: The account holder’s name.
  + Account account: The user's bank account.
* **Methods**:
  + Getters for all attributes: getUserId(), getUserPin(), getName(), getAccount().

**3.5 Account Class**

* **Responsibilities**: Manages the bank account transactions and maintains the account balance and transaction history.
* **Attributes**:
  + String accountNumber: A unique identifier for the account.
  + double balance: Stores the current balance of the account.
  + List<BankTransaction> transactionHistory: A list of transactions performed on the account.
* **Methods**:
  + getBalance(): Returns the current account balance.
  + deposit(double amount): Deposits the specified amount into the account and creates a new BankTransaction for the deposit.
  + withdraw(double amount): Withdraws the specified amount if the balance is sufficient and records a BankTransaction for the withdrawal.
  + transfer(Account targetAccount, double amount): Transfers the specified amount to another account.
  + getTransactionHistory(): Returns the list of all transactions associated with the account.

**3.6 BankTransaction Class**

* **Responsibilities**: Represents individual transactions, such as deposits, withdrawals, and transfers.
* **Attributes**:
  + String transactionId: A unique identifier for each transaction.
  + String transactionType: Describes the type of transaction (Deposit, Withdraw, Transfer).
  + double amount: The amount involved in the transaction.
  + Date date: The date and time of the transaction.
* **Methods**:
  + Constructor: Initializes the transactionType, amount, and date, and generates a unique transactionId.
  + Getters for transaction details: getTransactionId(), getTransactionType(), getAmount(), getDate().

**4. Detailed Flow of Operations**

1. **Starting the ATM System**:
   * The program begins in the main() method, where an instance of the Bank and ATM is created.
   * The ATM instance starts an infinite loop offering users the options to log in, create a new account, or exit.
2. **User Login**:
   * The user provides their user ID and PIN.
   * The ATM invokes the Bank's authenticateUser() method, which checks if the provided credentials match any account holder.
3. **Creating a New Account**:
   * The user is prompted to provide their name, user ID, PIN, and an initial deposit.
   * A new AccountHolder is created, along with a new Account, and both are stored in the Bank.
4. **Performing Transactions**:
   * Once logged in, users can choose from several options:
     + **Withdraw**: The user enters the amount to withdraw, which is deducted from their balance if sufficient funds are available.
     + **Deposit**: The user enters the amount to deposit, which is added to their account balance.
     + **Transfer**: The user provides the target account number and amount to transfer, which deducts from their balance and adds to the target account’s balance.
     + **Transaction History**: Displays a list of all transactions performed on the user's account.
5. **Transaction Recording**:
   * Each transaction (withdrawal, deposit, transfer) creates a new BankTransaction object, which is added to the account’s transactionHistory.

**5. Conclusion**

The ATM application is structured in a simple yet modular fashion, with clear separation of responsibilities between different classes. The ATM class serves as the controller, interacting with the user and invoking the appropriate methods on the Bank, Account, and AccountHolder classes to perform operations. Each transaction is recorded for future reference, ensuring a comprehensive transaction history.

This low-level design provides a clear understanding of how the system operates at a granular level, with each class and method serving a specific purpose within the ATM simulation.