**Documentation of The Banking System**

**Account Class**

Stores basic account info like bank details, account number, and owner details (name, email, PIN). Keeps a record of all past transactions

***Added Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *loadTransactionsFromDatabase()* | It clears the existing transaction list to avoid duplicates, fetches transactions from a log service using the account number, adds them back into the list, and prints the number of transactions loaded.  ***Why it was added:***  To keep an account’s transaction history updated by automatically retrieving past transactions from a database. This makes sure that it has accurate financial records without manual entry. |

**Business Account Class**

Gets twice the credit limit of a regular credit account. Can take and manage loans, but payments are only allowed to savings accounts. Supports loan repayments with higher limits and uses TransactionServices for secure transactions.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *TransactionServices transactionService* | Handles transaction operations. |
| *double BUSINESS\_CREDIT\_LIMIT\_MULTIPLIER* | Multiplier to increase credit limit for business accounts (default 2.0). |

***Methods***

| Method Name | Description |
| --- | --- |
| *getBusinessCreditLimit()* | Returns the total credit limit for the business account. |
| *canBusinessCredit(double amount)* | Checks if the requested credit is within the business credit limit.  Params: amount - The loan amount to be checked.  Returns: true if within the limit, otherwise false. |
| *pay(Account recipient, double amount)* | Business accounts can only pay Savings accounts. Uses TransactionServices for processing.  Params:  recipient - The receiving SavingsAccount.  amount - The payment amount.  Returns: true if the payment succeeds, otherwise false. |
| *recompense(double amount)* | Processes recompense payments via TransactionServices.  Params: amount - The repayment amount.  Returns: true if recompense is successful, otherwise false. |
| adjustLoanAmount(double amount) | Adjusts the loan balance but ensures it does not exceed the business credit limit.  Params:  amount - The amount to adjust (positive to increase, negative to decrease). |
| getLoanStatement() | Generates and returns a formatted loan statement for a business account.  Returns: A String containing the business account loan statement. |
| *toString()* | *Returns a formatted string representation of the business account details.* |

**Credit Account Class**

Lets users borrow money for transactions. Keeps track of outstanding loan balance, prevents overspending, and only allows payments to savings accounts. Supports loan repayments and provides loan statements.

***Added Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *pay()* | allows a CreditAccount to send money to a SavingsAccount by checking if the credit limit allows the transaction. If the limit is exceeded, the payment fails. Otherwise, it processes the transfer through transactionService.creditPayment().  ***Why it was added:***  To allow credit accounts to make controlled payments while ensuring they do not exceed their borrowing limit. |
| *recompense()* | It enables the CreditAccount to repay its loan balance by calling transactionService.recompense().  ***Why it was Added:***  To provide a way for users to settle their debts and restore available credit. |

**Student Account Class**

The StudentAccount class represents a specialized type of savings account designed for students.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *withdrawal(double amount)* | Withdraws an amount from the student account using TransactionServices, ensuring withdrawals adhere to account rules.  Params: amount - The amount to withdraw.  Returns: true if withdrawal is successful, otherwise false. |
| *transfer(Account recipient, double amount)* | Transfers funds to another SavingsAccount, ensuring the transaction does not exceed the student transfer limit (MAX\_WITHDRAWAL\_LIMIT).  Params:  recipient - The receiving account.  amount - The transfer amount.  Returns: true if the transfer is successful, otherwise false. |
| *getAccountBalanceStatement()* | Generates and returns a formatted balance statement for a student account.  Returns: A String containing the account balance statement. |
| *toString()* | Returns a formatted string representation of the student account details, including account number, owner name, bank, balance, and maximum withdrawal limit. |

**Savings Account Class**

Blocks invalid transactions (like sending money from one credit account to another). Shows clear error messages and ensures only valid transactions go through.

***Added Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *transfer(account: Account, amount: double)* | allows a SavingsAccount to send money to another account within the same bank. It calls transactionService.transferFunds() to process the transaction.  ***Why it was Added:***  It was added so users can transfer funds between accounts without manually withdrawing cash. |
| *transfer(bank: Bank, account: Account, amount: double)* | enables a SavingsAccount to send money to an account in a different bank, applying a processing fee. It also calls transactionService.transferFunds().  ***Why it was Added:***  We added it to support interbank transfers, allowing users to send money outside their bank while ensuring fees are applied. |
| *cashDeposit()* | allows a SavingsAccount to receive money by calling transactionService.deposit().  ***Why it was Added:***  So users can increase their account balance by depositing cash into their savings account. |
| *withdrawal()* | lets a SavingsAccount take out money by calling transactionService.withdraw().  ***Why it was Added:***  was added so users can access their funds when needed, ensuring they can withdraw cash from their account. |
| *getAccountBalance()* | returns the current balance of a SavingsAccount.  ***Why it was Added:***  was added to let users check how much money they have in their account. |

**Bank Class**

Handles everything related to banks, accounts, and transactions. Manages registered banks and tracks all transactions.

***Added Attributes:***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *String passcode* | Prevents unauthorized access to the bank’s data.  ***Why it was Added:***  is added to provide a security layer for the bank and to ensure only authorized users can access or modify bank details. |
| *String bankName* | Stores the name of the bank.  ***Why it was Added:***  Every bank has a unique name that helps in identification and differentiation from other banks in the system. This is useful when displaying bank details or performing searches. |
| *Int bankId* | A unique identifier is assigned to each bank.  ***Why it was Added:*** |
| *ArrayList<Account> bankAccounts* | A list that holds all the accounts associated with the bank.  ***Why it was Added:***  The bank needs to manage multiple accounts, so an ArrayList<Account> is used to store and retrieve accounts efficiently. It allows for operations like adding, searching, and displaying accounts under a specific bank. |
| *AccountService accountService* | A service instance that handles account-related operations, such as fetching and managing accounts from a database or external source.  ***Why it was Added:***  Instead of handling account retrieval and modifications directly within the Bank class, AccountService is used to separate concerns and improve modularity. |

***Added Methods:***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *loadAccountsFromDatabase()* | This method loads all accounts associated with the current bank from the database.  ***Why it was Added:***  This method ensures that account data is up-to-date by retrieving information from the database rather than relying on in-memory data. |
| *createNewStudentAccount()* | This method facilitates the creation of a new student account by collecting user inputs for account details such as account number, PIN, first name, last name, and email.  ***Why it was Added:***  Student accounts often have different policies (such as lower fees or special banking privileges), making it necessary to have a dedicated method for their creation. |
| *createNewBusinessAccount()* | This method is responsible for creating a new business account. It collects user input for the account number, PIN, name details, and email. Additionally, it asks for an initial loan amount, which is a distinguishing feature of business accounts.  ***Why it was Added:***  Business accounts often require additional financial features such as loans or credit lines. This method ensures proper handling of business-specific requirements while maintaining validation and consistency. |
| *showAccounts(Class<T> accountType)* | This method displays all accounts registered in the bank. It can either display all accounts or filter them based on the specified account type. If no accounts exist, it notifies the user.  ***Why it was Added:***  This method provides a structured way to review bank accounts, making it easier for administrators or users to check their accounts based on type. |
| *toString()* | This method provides a formatted string representation of the bank’s details, including its ID, name, passcode, deposit and withdrawal limits, credit limits, processing fee, and the number of registered accounts.  ***Why it was Added:***  This method makes it easier to log and display bank information in a structured format, which helps debug, report, and user interaction. |

**Bank Launcher Class**

Loads banks and accounts, helps create new ones, verifies bank credentials, and tracks transactions

***Added Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getBankByIndex()* | It checks if the given index is valid and returns the corresponding bank wrapped in an Optional. If the index is out of range, it returns an empty Optional.  ***Why it was added:***  This function simplifies retrieving a bank using an index, which is useful for menus where banks are listed numerically. |
| *getBanks()* | It returns the bank's list, allowing other parts of the program to access and manipulate the registered banks.  ***Why it was added:***  This function was added to provide access to the full list of registered banks, making it easier to retrieve all banks when needed. |

**Account Launcher Class** Manages login and authentication. Lets users pick a bank and account type and keeps track of active sessions.

***Added Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getLoggedAccount()* | Returns the currently logged-in account. |

**Business Account Launcher Class**

Gives business account users access to their menu, loan statements, payments, and transaction history. Ensures accounts are logged in before allowing transactions.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *businessAccountInit()* | Initializes the business account menu, allowing users to select options such as viewing loan statements, making payments, processing recompense transactions, and viewing transaction history. |
| *businessPaymentProcess()* | Handles the process of making payments from a business account to a savings account. The user enters a recipient account number and an amount, and the payment is processed if within credit limits. |
| *recompenseProcess()* | Manages the recompense process, allowing users to repay credit or loans. Ensures the recompense amount does not exceed the loan balance. |
| *getLoggedAccount()* | Retrieves the currently logged-in business account. Calls AccountLauncher.getLoggedAccount() and casts it to BusinessAccount. |

**Account Service Class**

Retrieves all accounts from the database, provides a static fetch method to access accounts without an instance, and allows account creation with validation. It also checks account balances, updates balances based on transactions, and manages loan updates for credit and loan accounts.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *IAccountDAO accountDAO* | Handles database operations for accounts. Used to fetch, update, and store account data. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *fetchAllAccounts()* | Gets a list of all accounts from the database. |
| *fetchAccountByNumber()* | Finds and returns an account using its account number.  Params:  number - Account number of the account to retrieve. |
| *fetchAllAccountsStatic()* | A static method to get all accounts without needing an instance of AccountService. |
| *createAccount(int, String, double, int, String, String, String, String)* | Creates a new account with basic validation.  Params:  bankId – ID of the bank where the account is created.  accountNumber – Unique account number.  balance – Initial balance of the account.  accountTypeId – Type of the account (e.g., savings, checking).  pin – Security PIN for account access.  ownerFname – First name of the account owner.  ownerLname – Last name of the account owner.  ownerEmail – Email of the account owner. |
| *getBalance(String)* | Returns the balance of a given account.  Params:  accountNumber – Account number to check balance for. |
| *updateBalance(Account, Double)* | Updates the balance of an account in the database.  Params:  account – The account to update.  amount – The new balance amount. |
| *updateLoan(Account, Double)* | Updates the loan balance of a credit account.  Params:  creditAccount – The loan or credit account to update.  amount – The amount to adjust the balance by. |

**BankService Class**

Manages banks by fetching, creating, and storing bank details in a database. It allows searching for banks by ID or name and supports setting custom transaction limits.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *IBankDAO bankDAO* | Handles database operations for banks, like fetching and storing bank details. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *fetchAllBanks()* | Retrieves all banks from the database. |
| *fetchAllBanksStatic()* | A static method to get all banks without needing an instance. |
| *fetchBankByID(int)* | Fetches a bank based on its unique identifier.  Params:  id - The bank ID.  Returns: A Bank object corresponding to the given ID. |
| *fetchBankByName(String)* | Fetches a bank based on its name.  Params:  name - The bank name.  Returns: A Bank object corresponding to the given name. |
| *createBank(int, String, String)* | Creates a new bank with basic details.  Params:  bankId - Unique identifier for the bank.  name - Name of the bank.  passcode - Security passcode for the bank. |
| *createBank(int, String, String, double, double, double, double)* | Creates a bank with custom limits for deposits, withdrawals, and fees.  Params:  bankId - Unique identifier for the bank.  name - Name of the bank.  passcode - Security passcode for the bank.  depositLimit - Maximum deposit allowed.  withdrawLimit - Maximum withdrawal allowed.  creditLimit - Maximum credit allowed.  processingFee - Transaction processing fee. |

**Log Service Class**

The LogService class provides functionalities for managing transaction logs within the banking system. It interacts with an ITransactionDAO implementation to fetch transaction records for a specific account and log new transactions.

***Attributes***

| ***Name and Data Type*** | ***Description*** |
| --- | --- |
| *ITransactionDAO* | Handles database operations related to transactions. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *fetchTransactionsForAccount()* | Retrieves all transactions for a given account  Params:  accountNumber – The unique number of the account. |
| *fetchTransaction()* | Fetches a specific transaction for an account  Params:  accountNumber – The unique number of the account. |
| *logTransaction()* | Records a new transaction with details like source, target, type, amount, and description.  Params:  sourceAccount – The account sending the money.  targetAccount – The account receiving the money.  type – The type of transaction (e.g., deposit, withdrawal).  amount – The transaction amount.  description – A brief note about the transaction. |

**Interfaces**

| ***Interface Name*** | ***Description*** |
| --- | --- |
| *Deposit* | It is an interface that serves as a blueprint for deposit operations |
| *FundTransfer* | It is an interface that defines the contract for transferring funds between bank accounts. It provides two methods to facilitate both intra-bank and inter-bank transfers while ensuring proper validation and error handling. |
| *Payment* | This interface defines a contact for processing payment between accounts in the banking system. |
| *Recompense* | This interface defines a mechanism for repaying a loan or reducing an outstanding credit balance in a banking system. |

**Security Util. Class**

The SecurityUtils class provides a utility method for hashing sensitive data, such as PINs, using the SHA-256 algorithm. This ensures secure storage and prevents plaintext exposure of sensitive information.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *hashCode(String pin)* | Hashes the provided PIN using the SHA-256 algorithm and returns the hashed value as a hexadecimal string.  Param:  pin – The plaintext PIN to be hashed. Return:  The hashed representation of the input PIN in hexadecimal format. |

**Service Provider Class**

Provides a centralized way to access essential services within the banking system. This class ensures that BankService, AccountService, and LogService are managed and retrieved efficiently through static methods.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *SQLiteDatabaseProvider databaseProvider* | Manages the database connection for data storage and retrieval. |
| *BankService bankService* | Provides bank-related functionalities, initialized with BankDAO. |
| *AccountService accountService* | Handles account-related operations, initialized with AccountDAO. |
| *LogService transactionService* | Manages transaction logs and records, initialized with TransactionDAO. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getBankService()* | Returns the singleton instance of BankService, used for bank-related operations. |
| *getAccountService()* | Returns the singleton instance of AccountService, which manages account creation, retrieval, and updates. |
| *getTransactionService()* | Returns the singleton instance of LogService, which logs and tracks transactions. |

**Transaction Class**

Provides a centralized way to access essential services within the banking system. This class ensures that BankService, AccountService, and LogService are managed and retrieved efficiently through static methods.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *String sourceAccount* | The account number that initiated the transaction. |
| *Transactions transactionType* | The type of transaction (e.g., deposit, withdrawal, transfer). |
| *String description* | A brief description of the transaction. |
| *LocalDateTime timestamp* | The timestamp indicating when the transaction was created. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getSourceAccount()* | *Returns the source account number that initiated the transaction.* |
| *getTransactionType()* | Returns the type of the transaction. |
| *getDescription()* | Returns the description of the transaction. |
| *getTimestamp()* | Returns the timestamp of when the transaction was created. |
| *toString()* | Returns a formatted string representation of the transaction, including time, source account, transaction type, and description. |

**Transaction Services Class**

The TransactionServices class provides various banking transaction functionalities, including fund transfers, cash deposits, withdrawals, credit payments, and loan repayments. It ensures account type validation, applies banking rules, and logs each transaction.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *transfer(Account sender, Account receiver, double amount)* | Transfers funds between two accounts within the same bank.  Params:   sender - Account initiating the transfer.  receiver - Account receiving the transfer.   amount - Amount to transfer.  Returns: true if successful, false otherwise.  Exceptions: IllegalAccountType if accounts are not SavingsAccount. |
| *transfer(Account sender, Bank receiverBank, Account receiver, double amount)* | Transfers funds between different banks.  Params:  sender - Account initiating the transfer.  receiverBank - Receiving bank.  receiver - Receiving account.  mount - Amount to transfer.  Returns: true if successful, false otherwise.  Excepations: IllegalAccountType if accounts are not SavingsAccount. |
| *cashDeposit(Account account, double amount)* | Handles cash deposits into a savings account.  Params:  account - Account receiving the deposit.  amount - Amount to deposit.  Returns: true if successful, false otherwise. |
| *withdraw(Account account, double amount)* | Withdraws cash from a savings account.  Params:  account - Account to withdraw from.  amount - Amount to withdraw.  Returns: true if successful, false otherwise. |
| *creditPayment(Account creditAcc, Account savingsAcc, double amount)* | Handles credit payments from a credit account to a savings account.  Params:  creditAcc - Credit account making the payment.  savingsAcc - Savings account receiving the payment.  amount - Amount to pay.  Returns: true if successful, false otherwise.  Exceptions: IllegalAccountType if account types do not match. |
| *recompense(Account creditAcc, double amount)* | Handles loan repayments for a credit account.  Params:  creditAcc - Credit account making the repayment.  amount - Amount to repay.  Returns: true if successful, false otherwise.  Exceptions: IllegalAccountType if account is not CreditAccount. |

**AccountDAO Class**

The AccountDAO class provides database access functionality for managing bank accounts. It interacts with the database to add, retrieve, and update account information.

***Attributes***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *IDatabaseProvider databaseProvider* | Provides the database connection for executing queries. This is injected through the constructor |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *addAccount(int bankId, String accountNumber, double balance, int accountTypeId, String pin, String ownerFname, String ownerLname, String ownerEmail)* | Adds a new account record to the database.  Params:  bankId - Identifier of the bank.  accountNumber - Unique account number.  balance - Initial account balance.  accountTypeId - Identifier for the account type.  pin - Security PIN for authentication.  ownerFname - First name of the account holder.  ownerLname - Last name of the account holder.  ownerEmail - Email address of the account holder. |
| *getBalance(String accountNumber)* | Retrieves the balance for a given account number.  Params:  accountNumber - The account number to check  balance for.  Returns: The balance of the account. |
| *updateBalance(String accountNumber, double newBalance)* | Updates the balance of an account in the database.  Params:  accountNumber - The account number to update.  newBalance - The new balance to set. |
| *getAllAccounts()* | Retrieves a list of all accounts stored in the database.  Returns:  A list of Account objects representing all accounts. |
| *getDBAccountByNumber(String accountNumber)* | Fetches a specific account by its account number from the database.  Params:  accountNumber - The account number to retrieve. Returns:  The Account object corresponds to the given account number. |

**AccountTypeDAO Class**

The AccountTypeDAO class provides database access functionality for managing account types. It allows inserting new account types and retrieving their corresponding IDs from the database.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *insertAccountType(String typeName)* | Inserts a new account type into the database. If the type already exists, the insertion is ignored.  Params:  typeName - Name of the account type to be inserted. |
| *getAccountTypeId(String typeName)* | Retrieves the unique identifier (ID) of a given account type.  Params:  typeName - The name of the account type to look up. Returns:  The ID of the account type if found; otherwise, returns -1. |

**BankDAO Class**

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *addBank(int bankId, String name, String passcode)* | Adds a new bank to the database with basic details.  Params:   bankId - Unique identifier for the bank. name - Name of the bank.  passcode - Security passcode for the bank. |
| *addBank(int bankId, String name, String passcode, double depositLimit, double withdrawLimit, double creditLimit, double processingFee)* | Adds a new bank with additional settings such as deposit, withdrawal, credit limits, and processing fees.  Params:   bankId - Unique identifier for the bank. name - Name of the bank.  passcode - Security passcode.   depositLimit - Maximum deposit allowed.   withdrawLimit - Maximum withdrawal allowed.   creditLimit - Maximum credit limit.   processingFee - Associated processing fees. |
| *getAllBanksFull()* | Retrieves a list of all banks stored in the database.  Returns:   A list of Bank objects representing all stored banks. |
| *getDBBankByID(int id)* | Retrieves a bank's details using its ID.  Params:   id - Unique identifier of the bank.  Returns:   A Bank object if found, otherwise null. |
| *getDBBankByName(String name)* | Retrieves a bank's details using its name.  Params:   name - Name of the bank.  Returns:   A Bank object if found, otherwise null. |

**DatabaseSchema Class**

This class handles the creation and clearing of tables in the database.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *createTables()* | Creates the necessary tables in the database if they do not exist.  Throws: SQLException if table creation fails. |
| *clearTables()* | Clears all data from the Transactions, Account, and Bank tables while maintaining the schema.  Throws: SQLException if clearing tables fails. |
| *clearAccountTypeTable()* | Clears all data from the AccountType table while maintaining the schema.  Throws: SQLException if clearing the table fails. |
| *main(String[] args)* | The main method for executing table creation. Uncomment the calls to clearAccountTypeTable() and clearTables() if you need to reset the database |

**DatabaseSeeder Class**

This class populates the database with initial data, including banks, account types, accounts, and sample transactions.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *insertBank(BankService bankService)* | Inserts predefined bank records into the database. |
| *insertAccountTypes(AccountTypeDAO accountTypeDAO)* | Inserts predefined account types into the database.. |
| *insertAccounts(AccountService accountService, AccountTypeDAO accountTypeDAO)* | Inserts sample accounts with associated account types and banks. |
| *insertSampleCreditPayments(TransactionDAO transactionDAO)* | Logs sample transactions including fund transfers, external transfers, credit payments, withdrawals, deposits, and compensations. |
| *insertSampleData()* | Calls the appropriate methods to insert sample banks, account types, accounts, and transactions into the database. |
| *main(String[] args)* | Entry point for executing the database seeding process. |

**IAccountDAO Interface**

This interface defines the operations for managing account data in the database.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *addAccount(int bankId, String accountNumber, double balance, int accountTypeId, String pin, String ownerFname, String ownerLname, String ownerEmail)* | Adds a new account to the database with the specified details. |
| *getBalance(String accountNumber)* | Retrieves the current balance of the specified account.  Returns: The account balance as a double. |
| *updateBalance(String accountNumber, double newBalance)* | Updates the balance of the specified account. |
| *getAllAccounts()* | Retrieves a list of all accounts in the database.  Returns: A List<Account> containing all account records. |
| *getDBAccountByNumber(String accountNumber)* | Retrieves the account details for the specified account number.  Returns: An Account object representing the account. |

**IBankDAO Interface**

This interface defines the operations for managing account data in the database.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *addBank(int bankId, String name, String passcode)* | Adds a new bank to the database with the specified ID, name, and passcode. |
| *addBank(int bankId, String name, String passcode, double depositLimit, double withdrawLimit, double creditLimit, double processingFee)* | Adds a bank with additional parameters, including transaction limits and processing fees. |
| *getAllBanksFull()* | Retrieves a list of all banks with complete details.  Returns: A List<Bank> containing all bank records. |
| *getDBBankByID(int id)* | Retrieves a bank record by its unique ID.  Returns: A Bank object representing the bank. |
| *getDBBankByName(String name)* | Retrieves a bank record by its name.  Returns: A Bank object representing the bank |

**IDatabaseProvider Interface**

This interface defines the contract for providing database connections. Implementing classes are responsible for establishing and managing database connections.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getConnection()* | Establishes and returns a database connection.  Returns: A Connection object for interacting with the database.  Throws: SQLException if the connection cannot be established. |

**ITransactionDAO Interface**

This interface defines the contract for handling transaction-related database operations. Implementing classes should provide functionality for logging transactions, retrieving transactions for specific accounts, and fetching transaction details.

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *logTransaction(String sourceAccount, String targetAccount, String type, double amount, String description)* | Logs a transaction in the database with details such as source and target accounts, transaction type, amount, and description. |
| *getTransactionsForAccount(String accountNumber)* | Retrieves a list of transactions associated with a specific account.  Returns: A List<Transaction> containing all transactions for the given account. |
| *getTransaction(String number)* | Fetches details of a specific transaction based on its unique identifier.  Returns: A String representation of the transaction details. |

**SQLiteDatabaseProvider Class**

This class provides a connection to an SQLite database. It implements the IDatabaseProvider interface and manages database connections using JDBC.

***Attribute***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *String DATABASE\_PATH* | Stores the JDBC connection string for the SQLite database. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *getConnection()* | Establishes and returns a connection to the SQLite database.  Returns: A Connection object to interact with the database.  Throws: SQLException if the connection cannot be established. |

**TransactionDAO Class**

This class handles database operations related to transactions. It implements the ITransactionDAO interface and provides methods to log transactions, retrieve transaction history for a specific account, and fetch transaction details.

***Attribute***

| ***Data Type and Name*** | ***Description*** |
| --- | --- |
| *IDatabaseProvider databaseProvider* | Handles database connections for executing SQL queries. |

***Methods***

| ***Method Name*** | ***Description*** |
| --- | --- |
| *logTransaction(String sourceAccount, String targetAccount, String type, double amount, String description)* | Logs a transaction by inserting a new record into the Transactions table. |
| *getTransactionsForAccount(String accountNumber)* | Retrieves a list of transactions associated with the specified account. It fetches transactions where the given account is either the source or target. |
| *getTransaction(String accountNumber)* | Returns a formatted string of all transactions for the specified account, including transaction type, amount, description, and source account. |