**Bank Management System Documentation**

**Class Descriptions with Method Parameters and Return Types**

**Transaction**

## Overview

The **Transaction** class in the OOP\_Project package is a crucial component of a sophisticated bank management system. It facilitates the handling of various financial transactions, including deposits, withdrawals, and transfers. This class incorporates robust exception handling to ensure the integrity of transactions and provides essential methods for reading and saving transactions.

* ***Purpose:*** Manages financial transactions.
* ***Attributes and Methods:***
* **transactionId (int):** Unique identifier for each transaction.
* **customerId (int):** Identifier of the customer initiating the transaction.
* **recipientId (int):** Identifier of the recipient in transfer transactions.
* **employeeId (int):** Identifier of the overseeing employee.
* **transactionStatus (boolean):** Flag indicating the success or failure of the transaction.
* **transactionDate (String):** Date and time when the transaction occurred.
* **transactionAmount (double):** Monetary value associated with the transaction.
* **transactionType (String):** Type of the transaction (e.g., deposit, withdrawal, transfer).
* **transactionDescription (String):** Additional information or reason associated with the transaction.
* **Constructor:**
* **Transaction (Deposit/Withdrawal):**
  + Handles deposit and withdrawal transactions.
  + Throws **TransactionException** for invalid or insufficient transactions.
* **Transaction (Transfer):**
  + Handles general transactions.
  + Throw transaction exceptions.
* **set...()** / **get...()**: Methods to set or get attributes. Return types vary based on attribute.
* toString(): Converts transaction information to a string. Returns String.
* **handleDeposit(Account account)throws TransactionException:** Handles deposit transactions by updating the account balance.
* **handleWithdrawal(Account account)throws TransactionException:** Handles withdrawal transactions by updating the account balance.

**Account (Abstract)**

* ***Purpose:*** Base class for different types of bank accounts.
* ***Attributes and Methods:***
  + accountNumber: int
  + balance: double
  + accountType :string
  + counter: static int
* **set...()** / **get...()**: Methods to set or get attributes. Return types vary based on attribute.
  + choice : int
  + accountLoan : Loan
  + **Account(double balance, String accountNumber)**: Constructor to initialize an account. No return.
  + **UpdateBalance(double amount)**: Update Balance.
  + **makeTransaction(int transactionType,ArrayList<Transaction> Alltransactions, ArrayList<Client> clients )throws TransactionException** : Processes a transaction. No return.
  + **checkSufficientBalance(double amount)**: Checks if the balance is sufficient for a transaction. Returns boolean.
  + **equals(Object obj)**: Compares with another object for equality. Returns boolean.
  + **display()**: Displays account details. No return.
  + **updateBalance(double amount)**: Updates the account balance. No return.
  + **takeLoan()** : Make Loan request no return.
  + payInstallment(): pay the instalment no return .
  + Account(int accountNumber , double balance ) : create account.

**SavingAccount**

* ***Inherits From:*** Account
* ***Attributes and Methods:***
  + **interestRate = 0.2 : Static double**
  + **accountDate : LocalDateTime**
  + **calculateInterest(LocalDateTime currentDate)**: Calculates interest for the savings account. No return.
  + **SavingAccount(double balance)**:create saving account

**CurrentAccount**

* ***Inherits From:*** Account
* ***Attributes and Methods:***
  + **MIN\_BALANCE = 3000 : static final double**
  + **Fees\_Rate = 0.1 : static final double**
  + **accountStatus : Boolean**
  + **CurrentAccount(double balance,ArrayList<transaction> allTransaction,int employeeId):** create current account and apply fees
  + **checkMinimumBalance()**: Checks if the current account has met the minimum balance requirement. Returns boolean.

**Client**

* ***Purpose:*** Manages information about bank clients.
* ***Attributes and Methods:***
  + id: String
  + firstName: String
  + lastName: String
  + username: String
  + telephoneNumber: long
  + myAccounts: ArrayList<Account>
  + **editPersonalInformation()**: Allows editing of client's personal information. No return.
* DisplayDetailsofhisAccount () No Return, this function allow client to display all his info
  + **readClient(ArrayList<Client> clients)**: Reads and displays client information. No return.
  + **passClient(ArrayList<Client> clients, ArrayList<Account> accounts)**: Passes client information for account creation. No return.

**Employee**

* ***Purpose:*** Manages employees within the bank system.
* ***Attributes and Methods:***
  + id: String
  + firstName: String
  + lastName: String
  + position: String
  + password : String
  + Total grade : String
  + Graduation collage : String
  + Year of graduation : Int
  + Status : Boolean
  + Address : String
  + **Employee(String password, String firstName, String lastName,** String address, String position, String graduatedCollage, int **yearOfGraduation, String totalGrade):**constructor used to create a new employee object
  + **EmployeeEditInfo():** This method is used when the employee wants to edit there address or position.
  + **EmployeeCreatingAccount(Arraylist<Client>clients2 , Int ID):** This method is used when the employee wants to create an account for a client.
  + **clientUPAuthentication(ArrayList<Client> clients, int id)**: Authenticates a client. Returns an object of client if the ID and password are correct otherwise null is returned.
  + **employeeSearchClient(ArrayList<Client> clients)**: Searches for a client and display there information.
  + **toString()**: Converts employee information to a string. Returns String.
  + **employeeGetInfo()**: Retrieves employee information. Returns String.
  + **createAClient(ArrayList<Client> clients)**: Creates a new client profile and adds this object to the clients array.
  + **readEmployee(ArrayList<Employee> employees)**: Reads employee information. No return.
  + **employeeDeletingAccount(ArrayList<Client> clients)**: Deletes a client account after getting his ID and password.
  + **EmployeeEditClient(ArrayList<Client>clients2 , int ID)**: This method is used when the employee wants to edit the username or password of a client , after getting his ID and password.
  + **EmployeeMakingTransaction(ArrayList<Client>clients , ArrayList<transaction>allTransaction)**: This method is used when the employee wants to transfer money from one account to another after making sure that the sender ID and password are correct.
  + **Save()** : returns a multiple line string used to write the data in the files in a certain format.
  + **SaveEmp(ArrayList<Employee>emps)** : This method loops on the array list of employees and calls the save method for each employee and writes the returned strings to the Employee file.
  + **readEmp(ArrayList<Employee>emps)**: This method is used to read the employees data from the files to the array list used in the program.
  + **Employee(String passw**
  + **ord,int id , String firstName, String lastName,** String address, String position, String graduatedCollage, int yearOfGradution , String totalGrade , Boolean status) : This constructor is used create an object employee and add it to the employee array list when reading from files.
  + **CreatingEmployeeAccount(ArrayList<Employee>employees):**This is a static method used when the user request to create an employee account. It asks for the user information and creates an employee object and adds it to the employees array list.
  + **toString():** overriding the toString() method to print the data of each employee.

**Loan**

* ***Purpose:*** Manages loans given to clients.
* ***Attributes and Methods:***
  + **loan(double amount, double balance)**: Constructor to initialize a loan. No return.
  + **clientBalance()**: Retrieves the balance of the client who took the loan. Returns double.
  + **displayLoanHistory()**: Displays the loan payment history. No return.
  + **payInstallment(int, boolean)**: Records a loan installment payment. No return.
  + **calculateInstallmentAmount()**: Calculates the amount for each installment. Returns double.
  + **initializeInstallments()**: Initializes the installment structure. No return.

**Installment**

* ***Purpose:*** Represents a single installment payment.
* ***Attributes and Methods:***
  + installmentNumber: int
  + installmentAmount: double
  + paid: boolean
  + **Installment(int number, double amount, boolean paid)**: Constructor to initialize an installment. No return.
  + **paidInstallment()**: Marks the installment as paid. No return.

**Admin**

* ***Purpose:*** Manages administrative tasks.
* ***Methods:***
  + **displayClients(ArrayList<Client> clients)**: Displays a list of clients. No return.
  + **displayEmployees(ArrayList<Employee> employees)**: Displays a list of employees. No return.
  + authorizeEmployee (**ArrayList<Employee> emps) : authorize employee**
  + **showTrancation(ArrayList<Employee> , ArrayList<Client> Clients , ArrayTranscation<Transaction> Transactions ) : show transactions**

**Main**

* ***Purpose:*** The entry point for the bank management system.
* ***Method:* main(String[] args)**: The main method to start the application. No return.

Main give you 4 options when you run the program

1-sign in as a client

2-sign as an employee

3-sign in as an admin

4-to save changes in files

If you choose 1:

it let you sign in and check if your ID and password is right then let you use client operations that we mentioned in client class if you press 3 it show you accounts (saving/current) that belong to your client then you can choose any account to access and make the operations on it and you can return back to your client operation and enter any other account

If you choose 2:

It let you sign in or sign up, if you choose to sign in it check if your ID and password is right then let you use employee operations that we mentioned in Employee class.

If you choose to sign up it let you create new employee

If you choose 3 it let you sign in as an admin it check if username and password right then show you admin operations

Every input you can return back from it

**TransactionException**

* ***Purpose:*** Custom exception class to handle transaction-related errors.