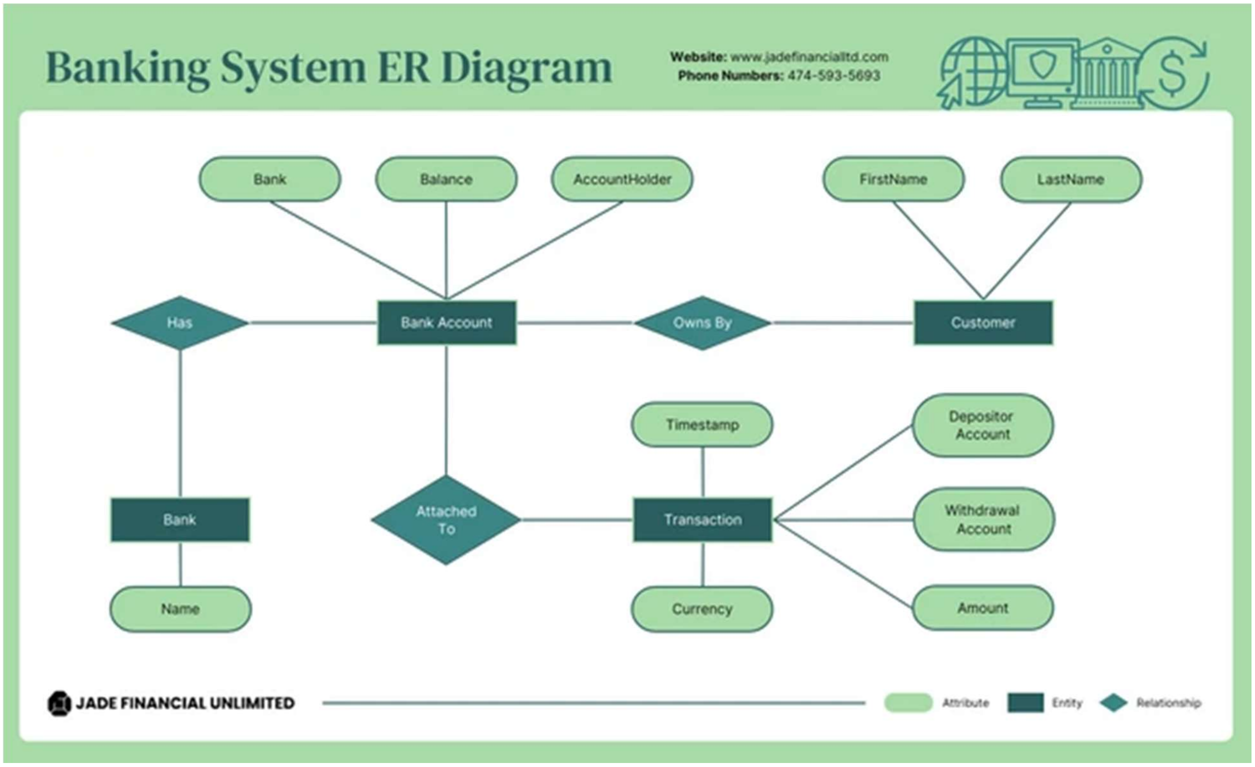

# Banking System ER Diagram

**Website:** www.jadefinancialltd.com
**Phone Numbers:** 474-593-5693

# Banking System Mini Project Documentation

This document provides a step-by-step guide for students to create a Banking System using Object-Oriented Programming (OOP) in Python. This project includes functionalities for users to perform banking operations and for the bank (admin) to manage accounts and view financial statistics.

## Objective

Create a Python-based banking system that:

1. Allows users to:
   - Open a new account.
   - Deposit money.
   - Withdraw money.
   - Check account balance.
   - Transfer money to another account.
   - View transaction history in a formatted statement.
2. Allows the bank (admin) to:
   - View total deposits in the bank.
   - Check the total number of accounts.

## Features and Functionality

**User Operations:**

1. **Open an Account**: Users can open a new account with a unique account number.
2. **Deposit Money**: Users can add money to their account balance.
3. **Withdraw Money**: Users can withdraw money, provided they have sufficient balance.
4. **Check Balance**: Users can view their current account balance.
5. **Transfer Money**: Users can transfer money to another existing account.
6. **Transaction Statement**: Users can view a detailed statement of all their transactions.

## Admin Operations:

1. **View Total Deposits**: Admins can see the total money deposited in the bank.
2. **Check Total Accounts**: Admins can see the total number of accounts in the bank.

# Implementation Steps

### Step 1: Define the BankAccount Class

The BankAccount class represents individual accounts and their operations.

**Attributes**:

- account_number: Unique account number for the account.
- account_holder: Name of the account holder.
- balance: Current balance in the account.
- transactions: A list to store the transaction history.

**Methods**:

- deposit(amount): Adds the specified amount to the account balance.
- withdraw(amount): Deducts the specified amount from the account balance if sufficient funds are available.
- check_balance(): Returns the current account balance.
- add_transaction(description): Adds a description of a transaction to the transaction history.
- print_statement(): Prints a detailed statement of all transactions.

### Step 2: Define the Bank Class

The Bank class manages all accounts and provides admin functionalities.

**Attributes**:

- accounts: A dictionary to store BankAccount objects, keyed by account numbers.

**Methods**:

- open_account(account_holder): Creates a new account for the specified account holder.
- get_account(account_number): Retrieves an account object using its account number.
- transfer(sender_account_number, receiver_account_number, amount): Transfers money between two accounts.
- admin_check_total_deposit(): Returns the total balance of all accounts in the bank.
- admin_check_total_accounts(): Returns the total number of accounts in the bank.

### Step 3: Create a Menu-Driven Interface

Provide an interactive menu to handle user and admin operations.

### Tips for Enhancement

1. Implement user authentication with a username and password.
2. Add account types (e.g., savings, current) with different features.
3. Include interest calculations for savings accounts.
4. Enhance the transaction history with timestamps.

## Submission:

**A well-structured python code on your Git hub account with all the functionality implemented.**