Bank System Code Documentation

Overview

The Bank System project is a simplified simulation of a banking environment that provides functionalities like user registration, account management, transaction history, loan processing, and more.

It is implemented in **Python** using **object-oriented programming (OOP)** principles.

Classes and Methods

1. Bank

Description:

Represents **the central banking system**, managing all users and their account information, as well as overall bank functions like loans and user management.

Attributes:

- **bankname:** Name of the bank.
- **branchbank:** Branch name.
- **bankbalance:** Bank's total balance.
- **AllowedAccountTypes:** Types of accounts supported.

```python
#The Bank class represents the banking system
class Bank:
  def __init__(self):
    self.bankname = "bank"
    self.branchbank = "Main Branch"
    self.bankbalance = 20000.0
    self.AllowedAccountTypes = ["Savings", "Checking", "Business", "Student"]
```

- **users_data:** Dictionary containing user credentials and profile data.

```
#Dictionary storing user credentials and profile info
self.users_data = {"johndoe":{
        "password": "12344567"
        ,"full name": "John Doe"
        ,"Account_type": "Savings"}

        ,"brucewayne":{
        "password": "98675342"
        ,"full name": "Bruce Wayne"
        ,"Account_type": "Checking",}

        ,"arthurmorgan":{
        "password": "78495162"
        ,"full name": "Arthur Morgan"
        ,"Account_type": "Business"}

        ,"walterwhite":{
        "password": "12344567"
        ,"full name": "Walter White"
        ,"Account_type": "Student"
        ,}}
```

- **accounts_data:** Dictionary holding account details.

```
#Dictionary storing full account data for users
self.accounts_data = {
        "johndoe": {
        "Account_holder": "John Doe",
        "Account_number": "1234-4567-8910-1112",
        "Account_type": "Savings",
        "Balance": 1000}

       ,"brucewayne": {
        "Account_holder": "Bruce Wayne",
        "Account_number": "9867-5342-1098-6753",
        "Account_type": "Checking",
        "Balance": 1500}

       ,"arthurmorgan": {
        "Account_holder": "Arthur Morgan",
        "Account_number": "7849-5162-3078-4951",
        "Account_type": "Business",
        "Balance": 2000}

       ,"walterwhite": {
        "Account_holder": "Walter White",
        "Account_number": "8452-0124-3487-0147",
        "Account_type": "Student",
        "Balance": 2500}}
```

Methods:

- **login (username, password):** Authenticates a user and returns account details if successful.

```
#Authenticate a user by verifying username and password.
def login(self, username, password):
  if username in self.users_data and self.users_data[username]["password"] == password:
    data = self.accounts_data[username]
    return data
```

- **add_user(user):** Adds a new user to the bank.
- **remove_user(user):** Removes a user from the bank system.

```
#Add a new user to the bank system.
def add_user(self, user):
  if user.username not in self.users_data:
    self.users_data[user.username] = {
                "password": user.password
               ,"full name" : user.full_name}

    print(f"\n[Bank] User {user.full_name} added.\n")
  else:
    print(f"\n[Bank] User '{user.full_name}' already exists.\n")

#Remove an existing user from the bank system
def remove_user(self, user):
  if user.username in self.users_data:
    del self.users_data[user.username]
    print(f"\n[Bank] User {user.full_name} removed.")
  else:
    print(f"\n[Bank] User '{user.full_name}' not found.")
```

- **show_all_users():** Displays all users registered in the bank.

```
#Display all users registered in the bank system.
def show_all_users(self):
  if not self.users_data:
    print("[Bank] No users found.")

  else:

    print("\n--------[Bank] List of all users:--------\n")
    for username, info in self.users_data.items():
      print(f" Name: {info['full name']}")
      print(f"Username: {username}")
      print("\n--------------------\n")
```

- **loan_request(loan_amount):** Deducts the requested loan amount from the bank's balance.
- **loan_settlement(loan_amount, settled_loan):** Adds the settled loan back to the bank's balance if fully paid.

```
#Deduct the loan amount from the bank's reserve if possible.
def loan_request(self, loan_amount):
  if loan_amount <= self.bankbalance:
    self.bankbalance -= loan_amount

#Add settled loan back to the bank balance if it's fully paid.
def loan_settlement(self, loan_amount, settled_loan):
  if settled_loan == loan_amount:
    self.bankbalance += settled_loan
```

2. User

Description:

Represents a user/customer of the bank.

Attributes:

- **full_name:** Full name of the user.
- **username:** Username.
- **password:** Password.
- **accounts:** Dictionary mapping account numbers to account objects.

```
#User class represents a bank customer.
class User:
  def __init__(self, full_name ,username, password):
    self.full_name = full_name
    self.username = username
    self.password = password
    self.accounts = {}
```

Methods:

- **change_password(input_username, current_password, new_password):** Changes the user's password after verification.

```
#Change user password after validation.
def change_password(self, input_username, current_password, new_password):
  if input_username != self.username:
    print("Incorrect username. Please try again.")
    return

  if current_password != self.password:
    print("\nIncorrect password. Please try again.")
    return

  if len(new_password) < 8:
    print("\nNew password must be at least 8 characters long.")
    return

  self.password = new_password
  print("\nPassword changed successfully.")
```

- **create_account(account):** Adds a new account to the user.
- **delete_account(account_number):** Removes an account from the user.

```
#Create a new bank account for the user.
def create_account(self, account):
  if account.Account_number in self.accounts:
    print(f"[Bank] Account with number '{account.Account_number}' already exists.")
  else:
    self.accounts[account.Account_number] = account
    print(f"[Bank] {account.Account_holder}'s account added.")

#Delete an account by account number.
def delete_account(self, account_number):
  if account_number in self.accounts:
    del self.accounts[account_number]
    print(f"[Bank] {account_number} account removed.")
  else:
    print(f"[Bank] Account number '{account_number}' not found.")
```

- **show_all_accounts():** Displays all accounts owned by the user.

```
    #Display all accounts owned by the user.
    def show_all_accounts(self):
      if not self.accounts:
        print("[Bank] No accounts found.")
      else:
        print("\n--------[Bank] List of all accounts:--------\n")
        for username, account in self.accounts.items():
          print(f"Name: {account.Account_holder}")
          print(f"Account Number: {account.Account_number}")
          print(f"Account Type: {account.Account_type}")
          print(f"Balance: ${account.Balance}\n")
          print("\n--------------------------------\n")
```

3. BankAccount (inherits from Bank)

Description:

Represents an individual bank account with its own transactions.

Attributes:

- **Account_holder:** Name of the account holder.
- **Account_type:** Type of account (Savings, Checking, etc.).
- **Balance:** Account balance.
- **Account_number:** Unique account number.
- **transactions:** List of transaction objects.

```
#BankAccount extends the Bank class and represents an individual account.
class BankAccount(Bank):
  def __init__(self, Account_holder,Account_type, Balance=0.0, Account_number=None):
    super().__init__()
    self.Account_holder = Account_holder
    self.Account_type = Account_type.strip().capitalize()
    self.Balance = Balance
    self.Account_number = Account_number or self.generate_account_number()
    self.transactions = []
```

Methods:

- **generate_account_number():** Creates a random formatted account number.

```
#Generate a random 16-digit formatted account number.
def generate_account_number(self):
  return f"{random.randint(1000, 9999)}-{random.randint(1000, 9999)}-{random.randint(1000, 9999)}-{random.randint(1000, 9999)}"
```

- **check_balance():** Displays the current balance.

```
#Check and print the current balance.
def check_balance(self):
  transaction = Transaction(
          transaction_type = "Checking Balance",
          transaction_amount="-",
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)
  self.transactions.append(transaction)
  print(f"Current balance: {self.Balance}\n")
```

- **Deposit(amount):** Deposits money into the account.

```
#Deposit a certain amount and log the transaction.
def Deposit(self, amount):
  self.Balance += amount
  transaction = Transaction(
          transaction_type = "Deposit",
          transaction_amount= amount,
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)

  self.transactions.append(transaction)
  print(f"Deposited: {amount}. New balance: {self.Balance}\n")
```

- **Withdraw(amount):** Withdraws money from the account.

```
#Withdraw funds if available and log the transaction.
def Withdraw(self, amount):
      if amount > self.Balance:
          print("\nInsufficient funds.")
      else:
          self.Balance -= amount
          print(f"Withdrew: {amount}. New balance: {self.Balance}\n")
          transaction = Transaction(
          transaction_type = "Withdraw",
          transaction_amount=amount,
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)

          self.transactions.append(transaction)
```

- **loan_request(loan_amount):** Adds loan amount to the account.

```
#Add loan amount to balance and log the transaction.
def loan_request(self, loan_amount):
      self.Balance += loan_amount

      transaction = Transaction(
          transaction_type = "Loan Request",
          transaction_amount=loan_amount,
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)

      self.transactions.append(transaction)
      print(f"Loan granted: {loan_amount}. New balance: {self.Balance}\n")
```

- **loan_settlement(loan_amount, settled_loan):** Deducts settled loan amount from balance.

```
#Deduct loan settlement from balance and log the transaction.
def loan_settlement(self, loan_amount, settled_loan):
      if settled_loan == loan_amount and settled_loan <= self.Balance:
          self.Balance -= settled_loan
          print(f"Loan of {loan_amount} settled.\n")
      else:
          print("Loan settlement failed.")

      transaction = Transaction(
          transaction_type = "Loan Settlement",
          transaction_amount=settled_loan,
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)

      self.transactions.append(transaction)
```

- **apply_interest(Rate_of_interest):** Applies interest to the account (only for Savings).

```
#Apply interest to balance for Savings accounts only.
def apply_interest(self, Rate_of_interest):
      if self.Account_type == "Savings":
          interest = self.Balance * Rate_of_interest
          self.Balance += interest
          print(f"Interest of {interest} applied.\n")

          transaction = Transaction(
          transaction_type = "Interest",
          transaction_amount= interest,
          Balance = self.Balance,
          date = datetime.date.today(),
          bankname= self.bankname,
          branchbank = self.branchbank,
          Account_holder= self.Account_holder,
          Account_number= self.Account_number)

          self.transactions.append(transaction)

      else:
          print("\nInterest applies only to Savings accounts.\n")
```

- **show_all_transactions():** Displays the transaction history.

```
#Display all transactions for this account.
def show_all_transactions(self):
  for trn in self.transactions:
    trn.transaction_details()
```

4. Transaction

Description:

Stores the details of a specific transaction.

Attributes:

- **transaction_type:** Type of transaction (Deposit, Withdraw, etc.).
- **transaction_amount:** Amount involved.
- **Balance:** Resulting balance.
- **date:** Date of transaction.
- **bankname, branchbank, Account_holder, Account_number:** Metadata.

```
#Transaction class stores details of a single transaction and provides a method to display them.
class Transaction:
  def __init__(self, transaction_type,transaction_amount, Balance, date, bankname, branchbank, Account_holder, Account_number):
    self.transaction_type = transaction_type
    self.transaction_amount = transaction_amount
    self.Balance = Balance
    self.date = date
    self.bankname = bankname
    self.branchbank = branchbank
    self.Account_holder = Account_holder
    self.Account_number = Account_number
```

Methods:

- **transaction_details():** Prints all information about the transaction.

```
#Print all details of a transaction.
def transaction_details(self):
  print("\n------------ Transaction -------------")
  print("type:", self.transaction_type)
  print("Amount:", self.transaction_amount)
  print("balance:", self.Balance)
  print("date:", datetime.date.today())
  print("bankname:", self.bankname)
  print("branch:", self.branchbank)
  print("account_holder:", self.Account_holder)
  print("account_number:", self.Account_number)
  print("---------------------------------------\n")
```

Example Scenarios from The Code:

1. Login and transactions:

```
#Entering correct user name and password.
Bank = Bank()
data1 = Bank.login("johndoe", "12344567")
if data1:
  print(f"Login successful. Welcome, {data1['Account_holder']}! \n")
  account1 = BankAccount(data1['Account_holder'],data1['Account_type'],data1['Balance'],data1['Account_number'])

  try:
    account1.check_balance()
    account1.Deposit(200)
    account1.Withdraw(200)
    account1.loan_request(400)
    account1.loan_settlement(400,400)
    account1.apply_interest(0.01)
    account1.show_all_transactions()

  except:
    print("Error Occurred")

else:
  print("Login failed! Invalid username or password.")
```

Output:

```
Login successful. Welcome, John Doe!

Current balance: 1000

Deposited: 200. New balance: 1200

Withdrew: 200. New balance: 1000

Loan granted: 400. New balance: 1400

Loan of 400 settled.

Interest of 10.0 applied.
```

```
------------ Transaction -------------
type: Checking Balance
Amount: -
balance: 1000
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

```
------------ Transaction -------------
type: Withdraw
Amount: 200
balance: 1000
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

```
------------ Transaction -------------
type: Loan Settlement
Amount: 400
balance: 1000
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

```
------------ Transaction -------------
type: Deposit
Amount: 200
balance: 1200
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

```
------------ Transaction -------------
type: Loan Request
Amount: 400
balance: 1400
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

```
------------ Transaction -------------
type: Interest
Amount: 10.0
balance: 1010.0
date: 2025-05-14
bankname: bank
branch: Main Branch
account_holder: John Doe
account_number: 1234-4567-8910-1112
---------------------------------------
```

2. Attempting to login with wrong credentials:

```
#Trying to login with wrong password.
data2 = Bank.login("brucewayne", "12344567")
if data2:
  print(f"Login successful. Welcome, {data2['Account_holder']}! \n")
  account2 = BankAccount(data2['Account_holder'],data2['Account_type'],data2['Balance'],data2['Account_number'])

  try:
    account1.check_balance()
  except:
    print("Error Occurred")

else:
  print("Login failed! Invalid username or password.")
```

```
Login failed! Invalid username or password.
```

3.Changing password:

```
#Changing password:

#Create a user with a username and a password.
user5 = User("Jane Austin" ,"janeaustin", "123456789")

#Call the change_password function
user5.change_password("johndoe", "12346789", "12345") #wrong username

user5.change_password("janeaustin", "12346789", "12345") #wrong password

user5.change_password("janeaustin", "123456789", "9876") #new password is less than 8 characters

user5.change_password("janeaustin", "123456789", "98765432") #password changed!
```

```
Incorrect username. Please try again.

Incorrect password. Please try again.

New password must be at least 8 characters long.

Password changed successfully.
```

4. Adding and removing users:

```
#Add,Remove and Display all users.
user5 = User("Alice Smith","alicesmith", "758410349")

Bank.add_user(user5)
Bank.show_all_users()

Bank.remove_user(user5)
Bank.show_all_users()
```

Output:

```
[Bank] User Alice Smith added.

--------[Bank] List of all users:--------

 Name: John Doe
Username: johndoe

--------------------

 Name: Bruce Wayne
Username: brucewayne

--------------------

 Name: Arthur Morgan
Username: arthurmorgan

--------------------

 Name: Walter White
Username: walterwhite

--------------------

 Name: Alice Smith
Username: alicesmith

--------------------
```

```
[Bank] User Alice Smith removed.

--------[Bank] List of all users:--------

 Name: John Doe
Username: johndoe

--------------------

 Name: Bruce Wayne
Username: brucewayne

--------------------

 Name: Arthur Morgan
Username: arthurmorgan

--------------------

 Name: Walter White
Username: walterwhite

--------------------
```

5. User managing their account:

```
#Creating,deleting an account for an existing user(Jane Austin) and showing all their accounts.
user5_acc1 = BankAccount("Jane Austin","Savings")

user5.create_account(user5_acc1)
user5.show_all_accounts()

user5.delete_account("7152-8652-9808-7588")
user5.show_all_accounts()
```

Output:

```
[Bank] Jane Austin's account added.

--------[Bank] List of all accounts:--------

Name: Jane Austin
Account Number: 7152-8652-9808-7588
Account Type: Savings
Balance: $0.0


--------------------------------

Name: Jane Austin
Account Number: 4860-5767-7981-2756
Account Type: Savings
Balance: $0.0


--------------------------------

[Bank] 7152-8652-9808-7588 account removed.

--------[Bank] List of all accounts:--------

Name: Jane Austin
Account Number: 4860-5767-7981-2756
Account Type: Savings
Balance: $0.0


--------------------------------
```