📓 Detailed Explanation of Python Banking System Code

1. Overview

- A simple, menu-driven banking system written in Python.
- Features include:
    - Opening accounts with minimum deposit rules.
    - Depositing and withdrawing money.
    - Checking balances.
    - Closing accounts.
    - Viewing all accounts.
    - Transaction history (audit trail).

---

2. Key Concepts

- Account: Holds customer details (name, city, balance, account number).
- Unique account number: Random 10-digit number, guaranteed unique.
- Minimum opening balance: INR 1000 required to open an account.
- Duplicate prevention: Stops multiple accounts for same name + city.
- Transaction history: Logs every action with timestamp.
- Menu system: User chooses actions by typing numbers (1–8).

---

3. How the System Works

1. Start menu: User sees options (Open, Deposit, Withdraw, etc.).
2. Open account:
    - Enter name, city, deposit ≥ 1000.
    - Confirm details before creation.
    - Account number generated, transaction logged.
3. Deposit/Withdraw:
    - Enter account number and amount.
    - Balance updated, transaction logged.
    - Withdrawals check for sufficient funds.
4. Balance enquiry: Shows account number and current balance.
5. Close account: Logs closure, removes account.
6. Display all accounts: Lists all active accounts.

7. Transaction history: Shows full audit trail.
8. Exit: Ends program.

---

4. Line-by-Line Code Walkthrough

Imports

- random: Used to generate random numbers.
- datetime: Used to timestamp each transaction.

```
import random
from datetime import datetime
```

Class setup

- Bank_Account: Defines what an account is and what it can do.
- accounts_list: Shared list of all active accounts.
- used_account_numbers: Shared set to remember which account numbers are already taken.

```
class Bank_Account:
    accounts_list = []    # shared list of all accounts
    used_account_numbers = set()   # to ensure uniqueness
```

Creating an account object

- init: Runs when a new account is created.
    - Customer_Name/City/Balance: Stores the user's details.
    - Customer_Account: Calls `generate_account_number()` to assign a unique 10-digit number.
    - transactions: A list to store audit history.
    - log_transaction("Account Opened", balance): Immediately logs the account creation.

```
def __init__(self, name, city, balance=0):
        self.Customer_Name = name
        self.Customer_City = city
        self.Customer_Balance = balance
        self.Customer_Account = self.generate_account_number()
        self.transactions = []  # transaction history list
        self.log_transaction("Account Opened", balance)
```

Generating a unique account number

- @classmethod: Method belongs to the class, not a single account.
- Loop: Keeps generating random 10-digit numbers until it finds one unused.
- Record usage: Adds the chosen number to `used_account_numbers` so it won't be reused.

```
@classmethod
    def generate_account_number(cls):
        """Generate unique 10-digit account number"""
        while True:
            acc_num = random.randint(10**9, 10**10 - 1)  # 10-digit
            if acc_num not in cls.used_account_numbers:
                cls.used_account_numbers.add(acc_num)
                return acc_num
```

Opening a new account

- Minimum balance check: Must be ≥ 1000; otherwise, prints a warning and stops.
- Duplicate check: If another account exists with the same name and city, cancels creation.
- Create and store: Makes a new `Bank_Account` and adds it to `accounts_list`.
- Print summary: Shows key details to the user.
- Return: Gives back the new account object (or `None` if failed).

```
@classmethod
    def open_account(cls, name, city, balance=1000):
        # Enforce minimum balance of 1000
        if balance < 1000:
            print("⚠ Minimum opening balance is INR 1000. Please
deposit at least 1000.")
            return None

        # Check duplicate (same name + city)
        for acc in cls.accounts_list:
            if acc.Customer_Name == name and acc.Customer_City ==
city:
                print("❌ Account already exists for this Name and
City.")
                return None

        new_acc = Bank_Account(name, city, balance)
        cls.accounts_list.append(new_acc)

        print(f"\n✅ Account opened successfully!")
        print("----- Account Information -----")
        print(f"Name       : {new_acc.Customer_Name}")
```

```
        print(f"City       : {new_acc.Customer_City}")
        print(f"Account No : {new_acc.Customer_Account}")
        print(f"Balance    : INR {new_acc.Customer_Balance}")
        print("-------------------------------")

        return new_acc
```

Logging a transaction (audit trail)

- Timestamp: Current date and time in "YYYY-MM-DD HH:MM:SS."
- Append record: Saves an entry with time, action type, amount, and balance after action.

```
def log_transaction(self, action, amount=0):
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.transactions.append(
            {"time": timestamp, "action": action, "amount": amount,
"balance": self.Customer_Balance}
        )
```

Deposit money

- Update balance: Adds the amount to current balance.
- Feedback: Prints confirmation with the new balance.
- Audit: Logs as "Deposit."

```
def deposit(self, amount):
        self.Customer_Balance += amount
        print(f"💰 Deposited INR {amount}. Updated Balance: INR
{self.Customer_Balance}")
        self.log_transaction("Deposit", amount)
```

Withdraw money

- Sufficient funds check: Only allows if `amount <= balance`.
- Update balance: Subtracts the amount and confirms.
- Audit: Logs "Withdrawal" or "Failed Withdrawal" accordingly.

```
def withdraw(self, amount):
        if amount <= self.Customer_Balance:
            self.Customer_Balance -= amount
            print(f"💸 Withdrawn INR {amount}. Updated Balance: INR
{self.Customer_Balance}")
            self.log_transaction("Withdrawal", amount)
        else:
```

```
            print("⚠ Withdrawal failed: Insufficient Balance.")
            self.log_transaction("Failed Withdrawal", amount)
```

Balance enquiry

- Display: Prints account number and current balance.
- Audit: Logs “Balance Enquiry.”

```
def balance_enquiry(self):
        print(f"📊 Account No: {self.Customer_Account}, Balance: INR
{self.Customer_Balance}")
        self.log_transaction("Balance Enquiry")
```

Close account

- Display: Shows closing confirmation and states payout method.
- Audit: Logs “Account Closed” with final balance.
- Cleanup: Removes the account from active lists and frees the account number.

```
def close_account(self):
        print(f"🔒 Account {self.Customer_Account} closed. Balance
INR {self.Customer_Balance} paid by cheque.")
        self.log_transaction("Account Closed",
self.Customer_Balance)
        Bank_Account.accounts_list.remove(self)

Bank_Account.used_account_numbers.remove(self.Customer_Account)
```

Show transactions

- Header: Indicates which account’s history is shown.
- Loop: Prints each entry with time, action, amount, and resulting balance.
- Footer: Separator line.

```
def show_transactions(self):
        print(f"\n📜 Transaction History for Account
{self.Customer_Account}:")
        for txn in self.transactions:
            print(f"{txn['time']} | {txn['action']} | Amount:
{txn['amount']} | Balance: {txn['balance']}")
        print("========================================")
```

Display all accounts

- Empty check: If no accounts exist, says so.
- List: Otherwise shows each account's name, city, number, and current balance.

```
@classmethod
    def display_all_accounts(cls):
        if not cls.accounts_list:
            print("🗁 No active accounts found.")
        else:
            print("\n===== Existing Accounts =====")
            for acc in cls.accounts_list:
                print(f"Name: {acc.Customer_Name}, City:
{acc.Customer_City}, "
                      f"Account No: {acc.Customer_Account}, Balance:
INR {acc.Customer_Balance}")
            print("============================")
```

The menu program

Find an account by number

- Search: Looks through `accounts_list` to find a matching account number.
- Return: Gives back the account object or `None`.

```
def find_account(acc_no):
    for acc in Bank_Account.accounts_list:
        if acc.Customer_Account == acc_no:
            return acc
    return None
```

Menu loop

- Infinite loop: Keeps showing options until you choose Exit (8).
- Choice input: Reads the user's selection.
- Branching: Runs the corresponding action.

```
def menu():
    while True:
        print("\n===== BANK MENU =====")
        print("1. Open Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Balance Enquiry")
        print("5. Close Account")
        print("6. Display All Accounts")
        print("7. Show Transaction History")
        print("8. Exit")
```

```
        choice = input("Enter your choice (1-8): ").strip()
```

Option 1: Open Account (with confirmation)

- Prompts: Name and city.
- Minimum deposit loop: Repeats until a valid numeric amount ≥ 1000 is entered.
- Confirmation screen: Shows summary and asks for Y/N.
- Create or cancel: Opens the account or cancels based on confirmation.

```
if choice == "1":
            print("\n--- Account Opening ---")
            print("Please provide the following details:")

            # Fresh prompts for customer details
            name = input("Customer Name: ").strip()
            city = input("Customer City: ").strip()

            # Keep prompting until minimum deposit of 1000 is
entered
            while True:
                try:
                    print("Opening Balance: ")
                    balance = float(input("Opening Balance (minimum
1000): ").strip())
                    if balance >= 1000:
                        break
                    else:
                        print("⚠ Minimum opening balance is INR
1000. Please try again.")
                except ValueError:
                    print("⚠ Invalid amount. Please enter a numeric
value.")

            # Confirmation summary
            print("\n--- Confirm Account Details ---")
            print(f"Name: {name}")
            print(f"City: {city}")
            print(f"Initial Deposit: INR {balance}")
            confirm = input("Confirm account opening? (Y/N):
").strip().lower()

            if confirm == "y":
                Bank_Account.open_account(name, city, balance)
            else:
                print("❌ Account opening cancelled.")
```

Option 2: Deposit

- Account lookup: Asks for account number and finds it.
- Amount: Reads deposit amount.
- Execute: Calls `deposit()` or shows “Account not found.”

```
elif choice == "2":
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                amount = float(input("Enter Deposit Amount:
").strip())
                acc.deposit(amount)
            else:
                print("❌ Account not found.")
```

Option 3: Withdraw

- Lookup: Same pattern as deposit.
- Amount: Reads withdrawal amount.
- Execute: Calls `withdraw()` which enforces sufficient funds.

```
elif choice == "3":
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                amount = float(input("Enter Withdrawal Amount:
").strip())
                acc.withdraw(amount)
            else:
                print("❌ Account not found.")
```

Option 4: Balance Enquiry

- Lookup: Finds account.
- Show balance: Calls `balance_enquiry()`.

```
elif choice == "4":
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                acc.balance_enquiry()
             else:
                 print("❌ Account not found.")
```

Option 5: Close Account

- Lookup: Finds account.
- Close: Calls `close_account()` which logs and removes it.

```
elif choice == "5":
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                acc.close_account()
            else:
                print("❌ Account not found.")
```

Option 6: Display All Accounts

- List all: Calls `display_all_accounts()`.

```
elif choice == "6":
            Bank_Account.display_all_accounts()
```

Option 7: Show Transaction History

- Lookup: Finds account.
- Show log: Calls `show_transactions()`.

```
elif choice == "7":
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                acc.show_transactions()
            else:
                print("❌ Account not found.")
```

Option 8: Exit

- Farewell: Prints a thank-you message and breaks the loop.

```
elif choice == "8":
             print("👏 Thank you for banking with us!")
             break
```

Invalid choice handler

- Fallback: Tells the user to try again.

```
else:
            print("⚠ Invalid choice. Please try again.")
```

Run the menu

- Start: Calls `menu()` to begin interaction.

```
# Run the menu
menu()
```

5. User Flows & Examples

- Open account: Enter name, city, deposit ≥ 1000 → confirm → account created.
- Deposit: Enter account number + amount → balance updated.
- Withdraw: Enter account number + amount → success if funds available, else failure.
- Balance enquiry: Shows account number + balance.
- Transaction history: Prints all actions with timestamps.
- Close account: Removes account, logs closure.

6. Safeguards & Improvements

Safeguards

- Minimum opening deposit enforced.
- Duplicate accounts prevented.
- Unique account numbers guaranteed.
- Audit trail for transparency.

Possible Improvements

- Validate all numeric inputs.
- Prevent negative/zero deposits/withdrawals.
- Add transfer feature between accounts.
- Save data to file/database for persistence.
- Add receipts/confirmation IDs.
- Role-based permissions (staff vs. customer).

7. Executable Python Code

```
# ------------------ MINI BANK ------------------

class Bank_Account:
    # Class variables (shared by all accounts)
    accounts_list = []    # Stores all account objects created
    used_account_numbers = set()  # Keeps track of account numbers to avoid
duplicates

    def __init__(self, name, city, balance=0):
        """
        Constructor method: runs automatically when a new Bank_Account
object is created.
        """
        self.Customer_Name = name           # Save customer's name
        self.Customer_City = city           # Save customer's city
        self.Customer_Balance = balance     # Save customer's balance
        self.Customer_Account = self.generate_account_number()  # Generate
unique account number
        self.transactions = []              # Empty list to store
transaction history
        self.log_transaction("Account Opened", balance)  # Record the
opening transaction

    @classmethod
    def generate_account_number(cls):
        """
        Generate a unique 10-digit account number.
        Uses random numbers and ensures no duplicates.
        """
        while True:  # Keep trying until a unique number is found
            acc_num = random.randint(10**9, 10**10 - 1)  # Random 10-digit
number
            if acc_num not in cls.used_account_numbers:  # Check uniqueness
                cls.used_account_numbers.add(acc_num)    # Reserve this
number
                return acc_num                           # Return it

    @classmethod
    def open_account(cls, name, city, balance=1000):
        """
        Open a new account with minimum deposit of 1000 INR.
        Prevents duplicate accounts (same name + city).
        """
        if balance < 1000:
            print("⚠ Minimum opening balance is INR 1000. Please deposit
at least 1000.")
            return None

        # Check if account already exists for same name + city
        for acc in cls.accounts_list:
            if acc.Customer_Name == name and acc.Customer_City == city:
                print("❌ Account already exists for this Name and City.")
                return None
```

```
        # Create new account object
        new_acc = Bank_Account(name, city, balance)
        cls.accounts_list.append(new_acc)  # Add to shared list

        # Print confirmation summary
        print(f"\n✅ Account opened successfully!")
        print("----- Account Information -----")
        print(f"Name       : {new_acc.Customer_Name}")
        print(f"City       : {new_acc.Customer_City}")
        print(f"Account No : {new_acc.Customer_Account}")
        print(f"Balance    : INR {new_acc.Customer_Balance}")
        print("-------------------------------")

        return new_acc

    def log_transaction(self, action, amount=0):
        """
        Record every transaction with timestamp, action, amount, and
balance.
        """
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # Current
time
        self.transactions.append({
            "time": timestamp,
            "action": action,
            "amount": amount,
            "balance": self.Customer_Balance
        })

    def deposit(self, amount):
        """
        Deposit money into account.
        """
        self.Customer_Balance += amount
        print(f"💰 Deposited INR {amount}. Updated Balance: INR
{self.Customer_Balance}")
        self.log_transaction("Deposit", amount)

    def withdraw(self, amount):
        """
        Withdraw money if balance is sufficient.
        """
        if amount <= self.Customer_Balance:
            self.Customer_Balance -= amount
            print(f"💸 Withdrawn INR {amount}. Updated Balance: INR
{self.Customer_Balance}")
            self.log_transaction("Withdrawal", amount)
        else:
            print("⚠️ Withdrawal failed: Insufficient Balance.")
            self.log_transaction("Failed Withdrawal", amount)

    def balance_enquiry(self):
```

```
        """
        Show current balance.
        """
        print(f"📊 Account No: {self.Customer_Account}, Balance: INR
{self.Customer_Balance}")
        self.log_transaction("Balance Enquiry")

    def close_account(self):
        """
        Close account and remove from system.
        """
        print(f"🔒 Account {self.Customer_Account} closed. Balance INR
{self.Customer_Balance} paid by cheque.")
        self.log_transaction("Account Closed", self.Customer_Balance)
        Bank_Account.accounts_list.remove(self)          # Remove from list
        Bank_Account.used_account_numbers.remove(self.Customer_Account)  #
Free account number

    def show_transactions(self):
        """
        Display all transactions for this account.
        """
        print(f"\n📜 Transaction History for Account
{self.Customer_Account}:")
        for txn in self.transactions:
            print(f"{txn['time']} | {txn['action']} | Amount:
{txn['amount']} | Balance: {txn['balance']}")
        print("============================================")

    @classmethod
    def display_all_accounts(cls):
        """
        Show all accounts currently open in the system.
        """
        if not cls.accounts_list:
            print("📂 No active accounts found.")
        else:
            print("\n===== Existing Accounts =====")
            for acc in cls.accounts_list:
                print(f"Name: {acc.Customer_Name}, City:
{acc.Customer_City}, "
                      f"Account No: {acc.Customer_Account}, Balance: INR
{acc.Customer_Balance}")
            print("============================")


# ----------------- HELPER FUNCTION -----------------

def find_account(acc_no):
    """
    Search for an account by account number.
    Returns account object if found, else None.
    """
```

```
    for acc in Bank_Account.accounts_list:
        if acc.Customer_Account == acc_no:
            return acc
    return None


# ------------------ MENU PROGRAM ------------------

def menu():
    """
    Interactive menu-driven program for banking operations.
    Keeps running until user chooses Exit.
    """
    while True:
        print("\n===== BANK MENU =====")
        print("1. Open Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Balance Enquiry")
        print("5. Close Account")
        print("6. Display All Accounts")
        print("7. Show Transaction History")
        print("8. Exit")

        choice = input("Enter your choice (1-8): ").strip()

        if choice == "1":
            # Account opening workflow
            print("\n--- Account Opening ---")
            name = input("Customer Name: ").strip()
            city = input("Customer City: ").strip()

            # Keep prompting until minimum deposit of 1000 is entered
            while True:
                try:
                    balance = float(input("Opening Balance (minimum 1000):
").strip())
                    if balance >= 1000:
                        break
                    else:
                        print("⚠ Minimum opening balance is INR 1000.
Please try again.")
                except ValueError:
                    print("⚠ Invalid amount. Please enter a numeric
value.")

            # Confirmation summary
            print("\n--- Confirm Account Details ---")
            print(f"Name: {name}")
            print(f"City: {city}")
            print(f"Initial Deposit: INR {balance}")
            confirm = input("Confirm account opening? (Y/N):
").strip().lower()
```

```
            if confirm == "y":
                Bank_Account.open_account(name, city, balance)
            else:
                print("❌ Account opening cancelled.")

        elif choice == "2":
            # Deposit workflow
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                amount = float(input("Enter Deposit Amount: ").strip())
                acc.deposit(amount)
            else:
                print("❌ Account not found.")

        elif choice == "3":
            # Withdrawal workflow
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                amount = float(input("Enter Withdrawal Amount: ").strip())
                acc.withdraw(amount)
            else:
                print("❌ Account not found.")

        elif choice == "4":
            # Balance enquiry workflow
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                acc.balance_enquiry()
            else:
                print("❌ Account not found.")

        elif choice == "5":
            # Close account workflow
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
                acc.close_account()
            else:
                print("❌ Account not found.")

        elif choice == "6":
            # Display all accounts
            Bank_Account.display_all_accounts()

        elif choice == "7":
            # Show transaction history
            acc_no = int(input("Enter Account Number: ").strip())
            acc = find_account(acc_no)
            if acc:
```

```
                acc.show_transactions()
            else:
                print("❌ Account not found.")

        elif choice == "8":
            # Exit program
            print("👋 Thank you for banking with us!")
            break

        else:
            print("⚠️ Invalid choice. Please try again.")


# ------------------ RUN THE PROGRAM ------------------
menu()
```

🏦 Understanding the Code: Bank Account Class + Menu Program

This code simulates a **mini banking system** in Python. It allows you to:

- Open accounts
- Deposit and withdraw money
- Check balances
- Close accounts
- View transaction history
- Run everything through a **menu-driven program**

---

1. Class Definition

```
class Bank_Account:
    accounts_list = []    # shared list of all accounts
    used_account_numbers = set()  # to ensure uniqueness
```

- `class Bank_Account:` → Defines a **blueprint** for creating bank account objects.
- `accounts_list = []` → A **class variable** (shared by all accounts). Stores all created accounts.
- `used_account_numbers = set()` → Another **class variable**. A `set` is used to store account numbers uniquely (no duplicates allowed).

---

2. Constructor (`__init__` method)

```
def __init__(self, name, city, balance=0):
    self.Customer_Name = name
    self.Customer_City = city
    self.Customer_Balance = balance
    self.Customer_Account = self.generate_account_number()
    self.transactions = []  # transaction history list
    self.log_transaction("Account Opened", balance)
```

- `__init__` → Special method called automatically when a new account is created.
- `self.Customer_Name` → Stores customer's name.
- `self.Customer_City` → Stores customer's city.
- `self.Customer_Balance` → Stores initial balance (default = 0).
- `self.Customer_Account` → Calls `generate_account_number()` to assign a unique 10-digit account number.
- `self.transactions = []` → Keeps a list of all transactions for this account.
- `self.log_transaction("Account Opened", balance)` → Logs the opening transaction.

---

3. Generating Account Numbers

```
@classmethod
def generate_account_number(cls):
    """Generate unique 10-digit account number"""
    while True:
        acc_num = random.randint(10**9, 10**10 - 1)  # 10-
digit
        if acc_num not in cls.used_account_numbers:
            cls.used_account_numbers.add(acc_num)
            return acc_num
```

- `@classmethod` → Method belongs to the class, not just one object.
- `random.randint(10**9, 10**10 - 1)` → Generates a random 10-digit number.
- `while True:` → Keeps looping until a unique number is found.
- `cls.used_account_numbers.add(acc_num)` → Adds the number to the set to prevent reuse.

---

4. Opening an Account

```
@classmethod
def open_account(cls, name, city, balance=1000):
```

```
    if balance < 1000:
        print("⚠ Minimum opening balance is INR 1000. Please
deposit at least 1000.")
        return None
```

- Enforces **minimum deposit of 1000 INR**.
- If balance < 1000 → Rejects account opening.

```
    for acc in cls.accounts_list:
        if acc.Customer_Name == name and acc.Customer_City ==
city:
            print("❌ Account already exists for this Name and
City.")
            return None
```

- Prevents **duplicate accounts** (same name + city).

```
    new_acc = Bank_Account(name, city, balance)
    cls.accounts_list.append(new_acc)
```

- Creates a new account object.
- Adds it to the shared `accounts_list`.

```
    print(f"\n✅ Account opened successfully!")
    ...
    return new_acc
```

- Prints account details.
- Returns the new account object.

---

5. Transaction Logging

```
def log_transaction(self, action, amount=0):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    self.transactions.append(
        {"time": timestamp, "action": action, "amount":
amount, "balance": self.Customer_Balance}
    )
```

- Records every action (Deposit, Withdrawal, Balance Enquiry, etc.).
- Stores a dictionary with:
    - `time`
    - `action`
    - `amount`
    - `balance`

---

6. Deposit & Withdraw

```
def deposit(self, amount):
    self.Customer_Balance += amount
    print(f"💰 Deposited INR {amount}. Updated Balance: INR
{self.Customer_Balance}")
    self.log_transaction("Deposit", amount)
```

- Adds money to balance.
- Logs the deposit.

```
def withdraw(self, amount):
    if amount <= self.Customer_Balance:
        self.Customer_Balance -= amount
        print(f"💸 Withdrawn INR {amount}. Updated Balance:
INR {self.Customer_Balance}")
        self.log_transaction("Withdrawal", amount)
    else:
        print("⚠ Withdrawal failed: Insufficient Balance.")
        self.log_transaction("Failed Withdrawal", amount)
```

- Checks if enough balance exists.
- Deducts money if possible, otherwise logs a failed withdrawal.

---

7. Balance Enquiry

```
def balance_enquiry(self):
    print(f"📊 Account No: {self.Customer_Account}, Balance:
INR {self.Customer_Balance}")
    self.log_transaction("Balance Enquiry")
```

- Prints current balance.
- Logs the enquiry.

---

8. Closing Account

```
def close_account(self):
    print(f"🔒 Account {self.Customer_Account} closed. Balance
INR {self.Customer_Balance} paid by cheque.")
```

```
    self.log_transaction("Account Closed",
self.Customer_Balance)
    Bank_Account.accounts_list.remove(self)

Bank_Account.used_account_numbers.remove(self.Customer_Account
)
```

- Prints closure message.
- Removes account from `accounts_list` and `used_account_numbers`.

---

9. Show Transactions

```
def show_transactions(self):
    print(f"\n📜 Transaction History for Account
{self.Customer_Account}:")
    for txn in self.transactions:
        print(f"{txn['time']} | {txn['action']} | Amount:
{txn['amount']} | Balance: {txn['balance']}")
    print("============================================")
```

- Loops through `transactions` list.
- Prints each transaction neatly.

---

10. Display All Accounts

```
@classmethod
def display_all_accounts(cls):
    if not cls.accounts_list:
        print("📂 No active accounts found.")
    else:
        print("\n===== Existing Accounts =====")
        for acc in cls.accounts_list:
            print(f"Name: {acc.Customer_Name}, City:
{acc.Customer_City}, "
                  f"Account No: {acc.Customer_Account},
Balance: INR {acc.Customer_Balance}")
        print("============================")
```

- Shows all accounts currently open.

---

11. Helper Function: Find Account

```
def find_account(acc_no):
    for acc in Bank_Account.accounts_list:
        if acc.Customer_Account == acc_no:
            return acc
    return None
```

- Searches for an account by account number.
- Returns the account object if found, else `None`.

---

12. Menu Program

```
def menu():
    while True:
        print("\n===== BANK MENU =====")
        ...
        choice = input("Enter your choice (1-8): ").strip()
```

- Runs an **infinite loop** until user chooses Exit.
- Displays menu options (Open, Deposit, Withdraw, etc.).
- Takes user input.

Each option (`if choice == "1": ... elif choice == "2": ...`) calls the corresponding method:

- `open_account`
- `deposit`
- `withdraw`
- `balance_enquiry`
- `close_account`
- `display_all_accounts`
- `show_transactions`
- `Exit (break)`

---

13. Run the Program

```
menu()
```

- Starts the banking system.