

VITYARTHI Project : MiniBank One

Introduction to Problem Solving and Programming

1. Project Title and Overview

- Project Title: **MiniBank One: Secure Banking System**
- Course Code/Name: **CSE 1021 - Introduction to Problem Solving and Programming**
- Name: **Ravi Prakash Rai**
- Reg. No.: **25BAI11199**
- Problem Statement: To design and implement a fundamental, secure, and persistent banking system that allows users to perform basic financial transactions (account creation, login, deposit, withdrawal, transfer) and view their complete transaction history.
- Technology Used: Python 3 (Standard Libraries Only)
- Design Philosophy: The system is implemented as a single-file Python script, leveraging **Object-Oriented Programming (OOP)** principles for account management and using **CSV files** for data persistence.

2. System Architecture and Design

2.1 Core Components

The system is structured around a single **Account** class and a main execution loop.

Component	Description
Account Class	Encapsulates all account data (account number, password, name, balance) and logic (deposit, withdraw, transfer, logging).
Data Persistence (CSV)	Each account's transaction history is stored in a dedicated file: <code>database/Statement_<account_no>.csv</code> . The current balance is derived from the last entry in this file.
Global Dictionary	The <code>accounts</code> dictionary in the main program stores all active Account objects in memory upon creation/login for quick lookup and operation.

2.2 Data Persistence Mechanism

The core requirement of data persistence is met using the built-in **csv** and **os** modules in Python.

1. A dedicated directory, **database**, is created to store all transaction files.
2. Each transaction (CREDIT or DEBIT) is logged into the respective account's CSV file, containing the fields: **Timestamp**, **Type**, **Amount**, **Balance**, and **Description**.
3. The **current balance** is not stored as a separate variable on disk. Instead, the `get_last_balance()` method retrieves the "**Balance**" value from the final row of the account's CSV file when the account is initialized or logged in.

3. Implementation Details

3.1 The Account Class

The `Account` class is the central unit of the system, demonstrating effective use of **encapsulation**.

Method	Purpose
<code>__init__(self, ...)</code>	Constructor. Initializes account details and checks for an existing transaction file. If the file exists, it loads the last balance; otherwise, it creates the file and logs the initial deposit.
<code>log(self, trans_type, amount, desc)</code>	Private utility method. Appends a new transaction record (including the updated balance) to the account's CSV file.
<code>get_last_balance(self)</code>	Reads the last recorded balance from the transaction CSV file to ensure persistent state across logins.
<code>deposit(self, amount)</code>	Adds the amount to the balance and logs a CREDIT transaction.
<code>withdraw(self, amount)</code>	Checks for sufficient balance, subtracts the amount, and logs a DEBIT transaction.

<code>transfer(self, to_acc, amount)</code>	Performs a debit from the source account and a simultaneous credit to the destination account, logging the corresponding transactions in <i>both</i> account files.
<code>show_statement(self)</code>	Reads and prints all transaction records from the account's CSV file.

3.2 Main Program Flow

The main program operates via a simple text-based menu-driven interface using a `while True` loop:

1. **Main Menu:** Offers **Create Account**, **Login**, and **Exit**.
2. **Account Creation (Ch 1):** Collects details and creates a new `Account` object, storing it in the global `accounts` dictionary.
3. **Login (Ch 2):** Authenticates the user based on account number and password.
4. **User Menu (Post-Login):** Offers **Deposit**, **Withdraw**, **Transfer**, **Statement**, and **Logout**. This nested loop manages post-login operations until the user logs out.

4. Key Programming Concepts Demonstrated

This project effectively demonstrates the following fundamental concepts of **Problem Solving and Programming**:

- **Object-Oriented Programming (OOP):** Use of the `Account class` to model a real-world entity, including **encapsulation** of data (balance, password) and behavior (deposit, withdraw).
- **Data Structures:** Use of a `dictionary (accounts)` for efficient storage and retrieval of active `Account` objects (mapping Account No. \rightarrow Account Object).
- **File Handling and Persistence:** Reading from and writing to **CSV files** using the `csv` module and managing directories with the `os` module. This solves the problem of retaining data after the program terminates.
- **Input/Output:** Implementation of an interactive **menu-driven interface** using `input()` and `print()`.
- **Flow Control:** Extensive use of `while` loops for continuous menu display and `if/elif/else` statements for decision-making and transaction validation (e.g., checking for insufficient balance).

5. Screenshots

1. Main

```
--- BANKING APP ---  
1 -> Create Account  
2 -> Login  
3 -> Close App
```

2. Create Account

```
--- BANKING APP ---  
1 -> Create Account  
2 -> Login  
3 -> Close App  
Choice: 1  
Enter Acc No: 1001  
Enter Name: Ravi Rai  
Enter Pass: 1234  
Enter Balance: 5000  
Account Ready!|
```

3. Login

a. Deposit

```
1 -> Create Account  
2 -> Login  
3 -> Close App  
Choice: 2  
Acc No: 1001  
Pass: 1234  
Login OK  
  
1.Deposit 2.Withdraw 3.Transfer 4.History 5.Logout  
Do: 1  
Amt: 500  
Money Added: 500.0  
Total: 5500.0
```

b. Withdraw

```
1.Deposit 2.Withdraw 3.Transfer 4.History 5.Logout
Do: 2
Amt: 200
Money Taken: 200.0
```

c. Transfer

```
1.Deposit 2.Withdraw 3.Transfer 4.History 5.Logout
Do: 3
To Acc: 1002
Amt: 600
Sent Successfully.
```

d. History

```
1.Deposit 2.Withdraw 3.Transfer 4.History 5.Logout
Do: 4
*****
USER: Ravi Rai
*****
['Date', 'Type', 'Amt', 'Bal', 'Info']
['2025-11-24', 'NEW', '5000.0', '5000.0', 'Open']
['2025-11-24', 'CR', '500.0', '5500.0', 'Deposit']
['2025-11-24', 'DR', '200.0', '5300.0', 'Withdraw']
*****
```

4. Logout

```
1 -> Create Account
2 -> Login
3 -> Close App
Choice: 3
Bye Bye
```

6. Conclusion and Learning Outcomes

MiniBank One is a successful implementation of a file-based banking system, achieving all stated features using only standard Python libraries.

- **Learning Outcomes:** The project provided practical experience in implementing **OOP principles**, handling **persistent data storage** via file I/O, and structuring a complex program using **modular code** and **menu navigation**.
- **Future Scope:** The system could be enhanced by implementing better security (hashing passwords), using a database (like SQLite) for more robust data management, and adding features like interest calculation or account closing.