Bank Management System – Learning Project Documentation

Author

Bhargav Ishaan

Language

Python

Project Type

Console-based, Object-Oriented Programming (OOP)

---

1. Project Overview

This project is a **console-based Bank Management System** built in Python to practice and understand:

- Object-Oriented Programming (OOP)
- Authentication logic
- Multi-user systems
- State management inside objects
- Menu-driven program design

The project was intentionally kept **non-GUI and non-database-based** to focus on **core logic and system thinking**, not frameworks.

---

2. Why This Project Was Built

At the time of building this project, I had:

- Finished Python basics

- Covered OOP concepts (classes, objects, `__init__`, methods)
- Learned inheritance and `super()`, but lacked confidence applying concepts

This project was chosen to:

- Convert theory into **practical instincts**
- Learn how real systems are structured
- Understand how multiple components interact in a program

---

3. Features Implemented

The final system supports:

- Creating **multiple bank accounts**
- Storing accounts using a **dictionary**
- Secure login using **account number + PIN**
- Limited PIN attempts (authentication logic)
- Viewing account details
- Depositing money
- Withdrawing money
- Maintaining **transaction history per account**
- Logout and session handling
- Menu-driven navigation (system menu + account menu)

---

4. System Design (High-Level)

Overall Structure

The program operates in **two levels**:

Level 1: System Menu

- Create new account
- Login to existing account
- Exit program

Level 2: Account Menu (after login)

- View account details
- Withdraw money
- Credit money
- View transaction history
- Logout

---

5. Core Data Structures Used

Dictionary for Account Storage

```
accounts = {
   account_number: BankObject
}
```

- Each account number maps to one `Bank` object
- This allows fast lookup and true multi-user behavior
- Acts as the **single source of truth**

List for Transaction History

- Each account maintains its own list of transactions
- Stored inside the account object
- Updated automatically during credit/withdraw operations

---

6. Bank Class Design

Attributes

- `name` – Account holder name
- `accno` – Account number
- `balance` – Current balance
- `pin` – Secret authentication PIN
- `transactions` – List of transaction records

Methods

- `verify_pin()` – Authenticates user
- `viewaccount()` – Displays account details
- `withdraw()` – Withdraws money with validation
- `credit()` – Credits money
- `show_transactions()` – Displays transaction history

---

7. Authentication Logic

- Login requires:
    - Account number
    - Correct PIN
- User is given **limited attempts**
- After failed attempts:
    - Login is denied
    - User must restart the login process

This taught:

- Separation of authentication from business logic
- Why input handling should stay outside classes

---

8. Key Concepts Learned

Python Concepts

- Classes and objects
- Instance variables
- Methods and `self`
- Dictionaries and lists
- Loops and conditional logic
- Input handling

OOP & Design Concepts

- Difference between **class vs object**
- Encapsulation of data
- Why logic belongs inside methods
- Why menus belong in the main program
- Single source of truth
- Session-based program flow

Real-World Thinking

- Banking workflow simulation
- Authentication flow design
- Multi-user system modeling
- State persistence in memory

---

9. Common Mistakes Faced (and Fixed)

- Calling methods on the **class** instead of the object

- Forgetting to update balance after crediting
- Using input inside class methods
- Placing transaction history in the wrong menu
- Loop conditions that never executed
- Creating duplicate objects instead of using dictionary references
- Typographical bugs causing logic errors

Fixing these mistakes contributed heavily to understanding **how programs actually execute**.

---

10. Why This Project Was Stopped

The project was intentionally stopped after:

- All learning goals were achieved
- Core system design was understood
- Further features would give diminishing learning returns

Stopping here avoided:

- Over-engineering
- Burnout
- Messy, duct-taped logic

---

11. Possible Future Improvements (Optional)

If revisited later, the project could be extended with:

- Saving accounts and transactions to a file
- Adding timestamps to transactions
- Permanent account lock after failed attempts
- Splitting into `BankAccount` and `BankSystem` classes
- GUI or web interface

---

12. Final Reflection

This project marked the transition from:

> “I know Python syntax”
> to
> **“I can design and reason about systems.”**

It served as a strong foundation for:

- Future Python projects
- DSA practice
- Backend development concepts
