Project Report: Python Bank Management System

Course : Python Programming

Student Name : Pranjal Singh

Registration No : 25BCE10746

Institute : VIT Bhopal

1. Introduction

This project implements a console-based Bank Management System in Python. It demonstrates fundamental Object-Oriented Programming (OOP) concepts such as classes, objects, encapsulation, and modular design while offering basic banking functionalities such as creating accounts, performing transactions, checking balances, and closing accounts.

Problem statement

There is a need for a simple and reliable system to manage basic banking operations such as account creation, deposits, withdrawals, and balance inquiries. Manual handling of these tasks is inefficient and increases the risk of errors. This project aims to develop a console-based Bank Management System using Python that automates these fundamental functions. The system utilizes Object-Oriented Programming principles to ensure modularity, accuracy, and ease of use while simulating real-world banking processes in a simplified environment.

Functional Requirements

1. Account Creation

- The system shall allow the user to create a new bank account.

- The system shall require the user to enter the account holder's name and initial deposit.
- The system shall automatically generate a unique 10-digit account number for each new account.

2. Deposit Operation

- The system shall allow the user to deposit money into an existing account.
- The system shall validate that the deposit amount is positive.
- The system shall update the account balance after a successful deposit.

3. Withdrawal Operation

- The system shall allow the user to withdraw money from an existing account.
- The system shall ensure that the withdrawal amount does not exceed the current balance.
- The system shall update the account balance after a successful withdrawal.

4. Balance Inquiry

- The system shall allow the user to check the current balance of an account.
- The system shall display the account number and corresponding balance.

5. View Account Details

- The system shall allow the user to view all details associated with an account.

- The system shall display the account holder’s name, account number, and current balance.

6. Account Search

- The system shall provide a method to locate and retrieve an account using its account number.

7. Account Closure

- The system shall allow the user to close an existing account.
- The system shall prompt the user for confirmation before permanently deleting the account.

8. Transaction Menu

- The system shall present a transaction menu to users accessing their accounts.
- The system shall allow navigation between deposit, withdrawal, balance inquiry, and detail viewing options.

9. Main Menu

- The system shall provide a main menu with options to create accounts, access existing accounts, close accounts, and exit the program.
- The system shall validate user inputs to avoid invalid menu selections.

10. Input Validation

- The system shall validate all numerical inputs to ensure they are valid numbers.
- The system shall display appropriate error messages when invalid data is entered.

Non-Functional Requirements

1. Usability

- The system shall provide a clear, easy-to-navigate text-based interface.
- The system shall present menu options in a simple and understandable format.
- The system shall display meaningful error messages for invalid inputs.

2. Reliability

- The system shall handle unexpected or invalid inputs without crashing.
- The system shall ensure accurate balance updates after each transaction.
- The system shall maintain consistent behaviour throughout all operations during execution.

3. Performance

- The system shall respond to user inputs within one second.
- The system shall process transactions (deposit, withdrawal, balance check) without noticeable delay.

4. Maintainability

- The system shall follow modular programming principles to simplify updates and modifications.

- The system's classes and methods shall be organized to support future enhancements such as transaction history or file storage.

5. Scalability

- The system shall support an increasing number of accounts without performance degradation during a single program session.
- The design shall allow future expansion, such as integration with databases or GUI interfaces.

6. Security

- The system shall prevent unauthorized access by requiring correct account numbers before performing transactions.
- The system shall not allow transactions involving negative or invalid amounts.

7. Portability

- The system shall run on any machine that supports Python without requiring platform-specific modifications.
- The system shall not depend on external frameworks beyond the standard Python library.

8. Data Integrity

- The system shall ensure that all balance calculations are precise and correctly updated.
- The system shall prevent inconsistent account states by validating all financial operations.

2.Objectives

1. To design a simple banking system using Python OOP principles.
2. To provide functionalities for deposit, withdrawal, and account management.
3. To demonstrate modular programming through the use of classes.
4. To simulate real-world banking operations in a simplified form.

3.System Overview

The system consists of two main classes: Account and Bank. The Account class handles individual account operations, while the Bank class manages multiple accounts and user interactions through menus and input handling.

3.1 Account Class

The Account class represents a single bank account with the following features:

1. Automatic 10-digit account number generation

2. Deposit functionality with validation

3. Withdrawal functionality with balance checking inquiry

4. Display of account details

3.2 Bank Class

The Bank class manages all customer accounts and high-level operations:

- Create new accounts
- Locate accounts via account number
- Close existing accounts
- Perform transactions such as deposit, withdraw, view details
- Main menu interaction loop

Development Process

4.1 Problem Definition

The task was to design a system that simulates basic banking operations in a way that is easy for beginners to understand while demonstrating core programming concepts.

4.2 Requirement Analysis

- Users should be able to create and access bank accounts.

- Basic financial transactions must be supported. Account details should be securely stored in memory during runtime.
- Input validation is required to avoid invalid transactions.

4.3 Top-Down Design/Modularization

The system is divided into separate modules: Account for individual user actions and Bank for overall system management.

4.4 Algorithm Development

Algorithms were developed for account creation, transaction handling, and user menu navigation, ensuring smooth operation and error handling

4.5 Implementation

The entire system is implemented in Python using OOP concepts. Dictionary-based data storage is used to map account numbers to account objects.

4.6 Testing & Refinement

Multiple test cases were executed to ensure accurate deposit, withdrawal, and balance operations. Input errors were handled using exception blocks

OUTPUT SCREENSHOTS

```
===== Welcome to Python Bank =====
1. Create New Account
2. Access Existing Account
3. Close an Account
4. Exit
Enter your choice (1-4): 2
Enter your account number: 9240405538

Welcome, Pranjal!
```

```
===== Welcome to Python Bank =====
1. Create New Account
2. Access Existing Account
3. Close an Account
4. Exit
Enter your choice (1-4): 1
Enter account holder's name: Pranjal
Enter initial deposit amount: $2340
Account for 'Pranjal' created successfully.
Your Account Number is: 9240405538
```

```
Welcome, Pranjal!

Transaction Menu:
1. Deposit
2. Withdraw
3. Check Balance
4. View Account Details
5. Return to Main Menu
Enter your choice (1-5): 2
Enter amount to withdraw: $345
Successfully withdrew $345.00. New balance: $1995.00
```

PROBLEM FACED

Problems Faced During Development

1. Input Validation Issues

Ensuring the user enters valid numerical values for deposits and withdrawals required additional error handling.

Incorrect or unexpected input types (e.g., letters where numbers are expected) caused early execution errors that needed to be resolved.

2. Unique Account Number Generation

Designing a method to automatically generate unique 10-digit account numbers without duplication presented an initial challenge.

Ensuring account numbers remained consistent across the program session required proper storage and referencing.

3. Managing Multiple Accounts

Storing and retrieving multiple accounts efficiently required creating a structured approach using lists or dictionaries.

Searching for specific accounts based on account numbers needed careful implementation to avoid incorrect account access.

4. Preventing Invalid Transactions

Implementing checks to prevent negative deposits, negative withdrawals, and overdrawing accounts required additional logical conditions.

Handling edge cases such as zero withdrawals or extremely large values needed to be addressed.

5. Menu Flow and Navigation

Designing a menu system that is intuitive and avoids user confusion was a challenge.

Ensuring the program returns to the correct menu after each operation needed careful flow control.

6. Ensuring Data Integrity

Balancing account updates during deposits and withdrawals needed precise logic to avoid incorrect balance calculations.

Preventing partial updates or inconsistent states required additional validation steps.

7. Program Termination and Error Handling

Ensuring the program does not crash when users enter invalid menu choices or abruptly exit operations required improved error handling.

Maintaining a smooth user experience even when errors occur was a major focus

LEARNINGS AND TAKEAWYAS

- Gained practical experience in implementing object-oriented programming concepts.
- Learned how to design modular, reusable classes for real-world applications.
- Improved skills in handling user input validation and error management.
- Understood how to manage data using dictionaries for efficient account retrieval.
- Enhanced problem-solving ability through designing and debugging a complete system.

FUTURE ENHANCEMENTS

- Implement a secure login system with PIN or password protection.
- Add persistent data storage using files or databases.
- Introduce online banking features such as fund transfers between accounts.
- Provide a graphical user interface (GUI) for improved usability.
- Include automated account statements and transaction history tracking.
- Add limits and notifications for large or unusual transactions.

REFERENCES

- W3Schools. Python Tutorials – Object-Oriented Programming, Functions, and File Handling.
- GeeksforGeeks. Python Programming Concepts and Examples.
- Stack Overflow Community – Assistance with debugging and best coding practices.
- Class lecture notes and course materials related to Python programming fundamentals.