Banking Management System

Submitted By

Student Name	Student ID
Mehedi Hasan Hira	0242220005101742
Md. Hafijur Rahman	0242220005101620
Afjal	0242220005101730

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course
**CSE 222: Object Oriented Programming II Lab
in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

November 2, 2024

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Nasima Islam Bithi**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

_____
Nasima Islam Bithi
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

_____
Mehedi Hasan Hira
ID:0242220005101742
Dept. of CSE, DIU

_____
Md. Hafijur Rahman
ID:0242220005101620
Dept. of CSE, DIU

_____
Afjal
ID:0242220005101730
Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:

Table 1: Course Outcome Statements

CO's	Statements
CO1	**Define** and **Relate** classes, objects, members of the class, and relationships among them needed for solving specific problems
CO2	**Formulate** knowledge of object-oriented programming and Python in problem solving
CO3	**Analyze** Unified Modeling Language (UML) models to **Present** a specific problem
CO4	**Develop** solutions for real-world complex problems **applying** OOP concepts while evaluating their effectiveness based on industry standards.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

The introduction chapter provides an overview of the Banking Management System, including its purpose, objectives, and scope. It outlines the significance of the project in enhancing banking operations and introduces the key features implemented using OOP concepts.

1.1   Introduction

Banking is an essential aspect of daily life that requires efficient systems for handling operations like deposits, withdrawals, and loan management. Manual banking systems often lead to operational inefficiencies, errors, and increased customer dissatisfaction. This project introduces a bank management system leveraging object-oriented programming (OOP) concepts to address these challenges by providing a modular and user-friendly solution. (Lutz, 2009)

1.2   Motivation

Managing banking systems manually can lead to errors, inefficiencies, and customer dissatisfaction. Automating banking operations like deposits, withdrawals, and loan management enhances customer experience and operational efficiency.

1.3   Objectives

- Implement a bank management system using object-oriented programming concepts.

- Develop a user-friendly menu-driven application for account management.

- Incorporate exception handling and ensure input validation.

- Demonstrate abstraction and multiple inheritance in the design.

1.4   Feasibility Study

The feasibility study evaluates the technical, operational, and economic aspects of existing hospital management systems to identify key limitations and areas for improvement. It provides insights into the gaps present in manual and basic digital systems, highlighting the need for a more integrated and scalable solution. (Lutz, 2009)

Feasibility Aspect	Case Study 1: Bank Management System
Technical Feasibility	The system is implemented in Python, which supports OOP principles and ensures scalability. Existing banking systems lack modularity and educational focus.
Gap Identified	Existing solutions do not fully leverage OOP principles like abstraction and inheritance for educational purposes. This system fills that gap.
Operational Feasibility	The system is designed to be a menu-driven application, making it easy to operate even for non-technical users.
Gap Identified	Manual banking processes can lead to delays and inefficiencies. Automation and a modular design address these issues.

1.5   Gap Analysis

Existing solutions often lack modularity or fail to integrate core OOP principles like abstraction and polymorphism in their design. This project aims to fill that gap with an educational focus.

1.6   Project Outcome

The project delivers the following outcomes:

- **Practical Application of OOP Principles:** Demonstrates the application of object-oriented programming concepts such as encapsulation, inheritance, abstraction, and polymorphism in a real-world scenario.

- **Efficient Banking Operations:** Automates core banking tasks such as deposits, withdrawals, balance inquiries, and notifications, reducing human error and operational inefficiencies.

- **Robust Exception Handling:** Ensures input validation and error management, enhancing system reliability.

- **Foundation for Future Enhancements:** Offers a platform for integrating advanced features such as GUI interfaces, multi-user support, and database integration in subsequent iterations.

Chapter 2

Proposed Methodology/Architecture

The Proposed Methodology/Architecture chapter will outline the system design, including the technologies used, database structure, and the workflow for implementing the Banking Management System.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The following steps outline the project approach:

1. Idea Selection

- We began by brainstorming ideas for a meaningful and practical project that could address real-world problems.

- After thorough discussions, we decided on developing a Bank Management System to streamline operations and improve user satisfaction.

2. System Design:

Class Design:

- Account (Base Class)

- SavingsAccount and CurrentAccount (Derived Classes)

- LoanAccount for loan management

- Notification (Multiple Inheritance Example)

- Abstract Base Class for enforcing core methods like create_account() and transaction()

3. Technical Requirements:

- Python Programming Language

- IDE: PyCharm/VSCode

- OOP principles: Abstraction, Encapsulation, Inheritance, Polymorphism

4. Implementation Using Python

- The system was developed in Python using OOP concepts such as classes, inheritance, and abstraction.

- Exception handling was implemented to ensure robust input validation and error management.

- Sample data and test cases were prepared to validate system functionality.

5. Testing the System

- Rigorous testing was conducted to identify and fix errors in the system design and implementation.

- Test cases covered all features such as account creation, deposit, withdrawal, and loan management.

6. Finalizing the Project

- After testing, we refined the system by improving the code structure, optimizing operations, and ensuring all features worked seamlessly.

- A flowchart was created to represent the system workflow visually, aiding in better understanding and presentation.
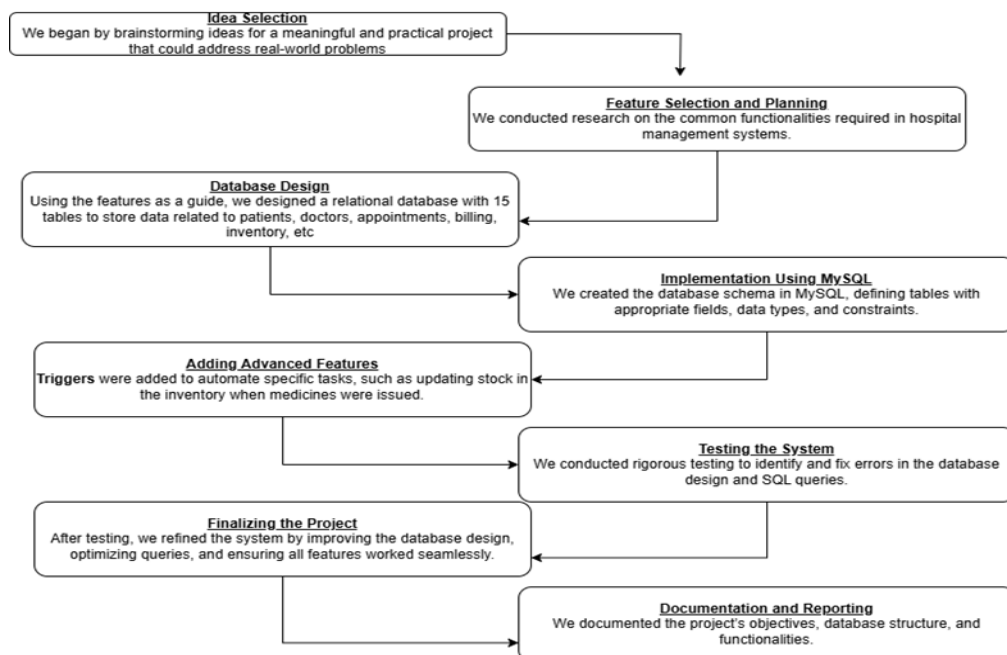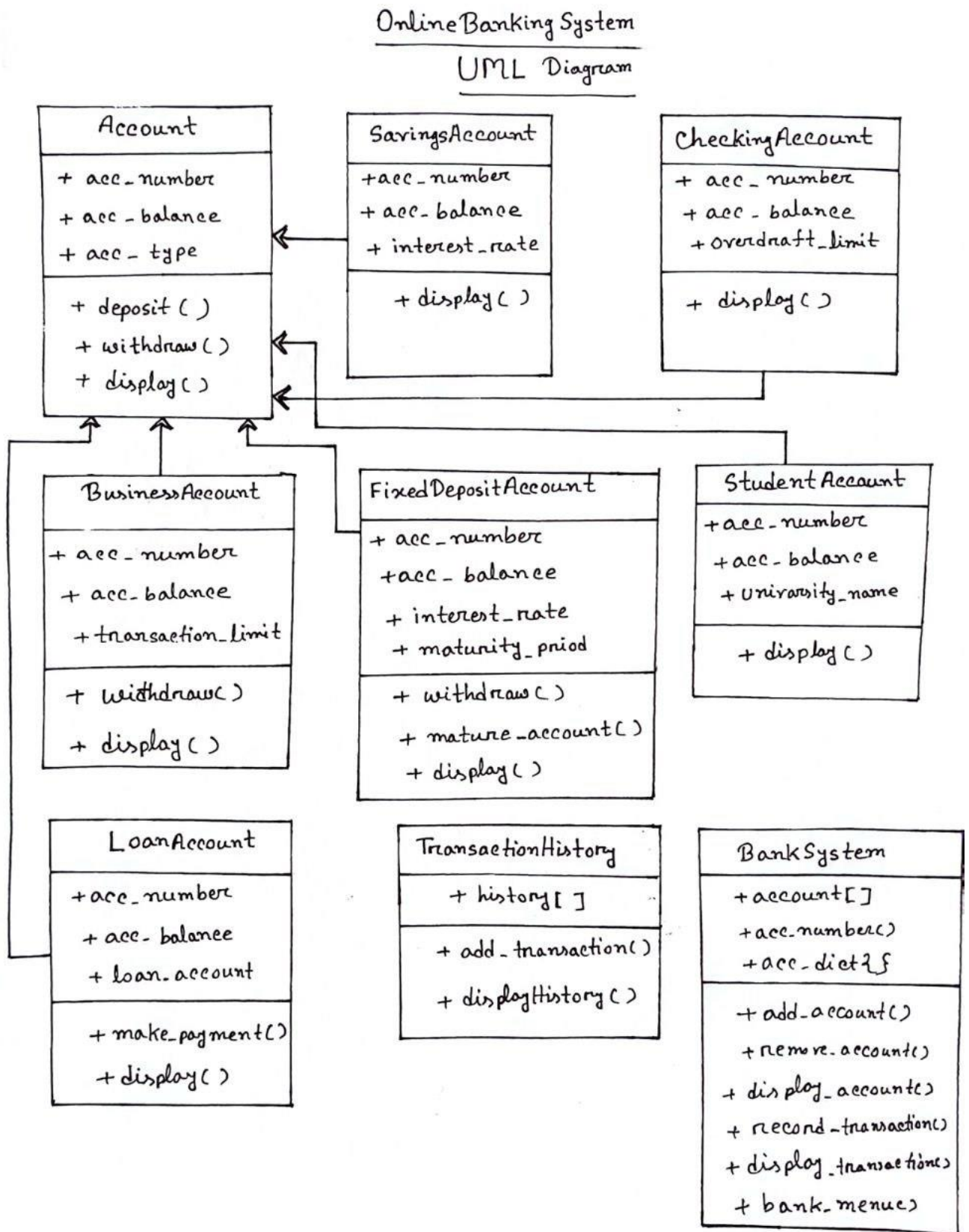
2.1.1 Proposed Methodology/ System Design

Idea Selection
We began by brainstorming ideas for a meaningful and practical project that could address real-world problems

Feature Selection and Planning
We conducted research on the common functionalities required in hospital management systems.

Database Design
Using the features as a guide, we designed a relational database with 15 tables to store data related to patients, doctors, appointments, billing, inventory, etc

Implementation Using MySQL
We created the database schema in MySQL, defining tables with appropriate fields, data types, and constraints.

Adding Advanced Features
Triggers were added to automate specific tasks, such as updating stock in the inventory when medicines were issued.

Testing the System
We conducted rigorous testing to identify and fix errors in the database design and SQL queries.

Finalizing the Project
After testing, we refined the system by improving the database design, optimizing queries, and ensuring all features worked seamlessly.

Documentation and Reporting
We documented the project's objectives, database structure, and functionalities.

Figure 2.1: This is a system diagram

Online Banking System

UML Diagram

Account

+ acc-number
+ acc-balance
+ acc-type

+ deposit()
+ withdraw()
+ display()

SavingsAccount

+ acc-number
+ acc-balance
+ interest_rate

+ display()

CheckingAccount

+ acc-number
+ acc-balance
+ overdraft_limit

+ display()

BusinessAccount

+ acc-number
+ acc-balance
+ transaction_limit

+ withdraw()
+ display()

FixedDepositAccount

+ acc-number
+ acc-balance
+ interest_rate
+ maturity_priod

+ withdraw()
+ mature-account()
+ display()

StudentAccount

+ acc-number
+ acc-balance
+ university_name

+ display()

LoanAccount

+ acc-number
+ acc-balance
+ loan-account

+ make_payment()
+ display()

TransactionHistory

+ history[ ]

+ add-transaction()
+ displayHistory()

BankSystem

+ account[]
+ acc.number()
+ acc-dict{}

+ add_account()
+ remove-account()
+ display_account()
+ record-transaction()
+ display-transaction()
+ bank-menue()

2.2   Overall Project Plan

The project was designed and completed within a span of 20 days, with each phase focused on specific tasks: **(Lutz, 2009)**

Phase 1: Planning and Requirement Analysis (Days 1–3)

- Finalizing the project idea.
- Identifying key features.
- Drafting the initial workflow and requirements

Phase 2: Database Design (Days 4–7)

- Creating ERD and designing tables.
- Defining relationships and constraints.
- Implementing and optimizing the database schema.

Phase 3: System Development (Days 8–12)

- Writing Python classes and implementing core functionalities.
- Adding exception handling and validations.
- Testing modules and fixing issues.

Phase 4: System Testing and Validation (Days 13–16)

- Testing individual features and ensuring robustness.
- Fixing bugs and optimizing performance.

Phase 5: Finalization and Documentation (Days 17–20)

- Preparing diagrams and documentation.
- Compiling the project report.
- Reviewing and preparing for a presentation.

Chapter 3
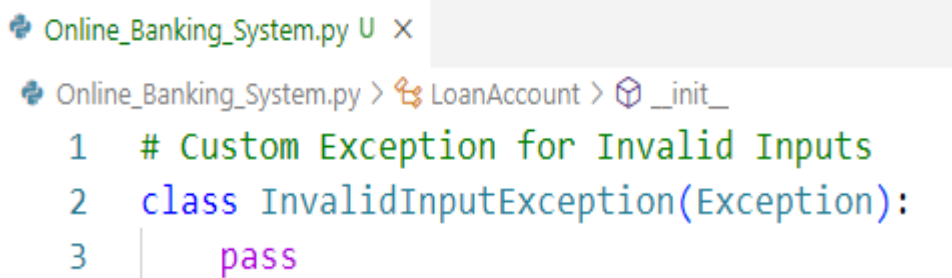
Implementation and Results

The "Implementation and Results" chapter will detail the step-by-step process of system development, including class creation, feature implementation, and a summary of results validated through testing.

3.1 Implementation
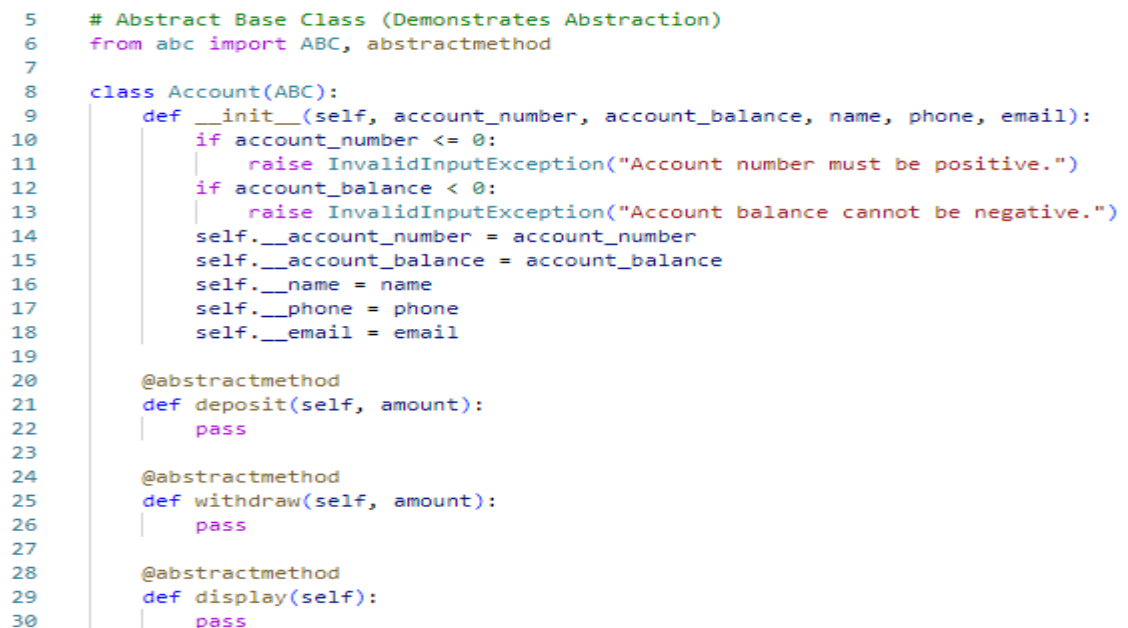The project is implemented using Python. Key features include:

1.Custom Exceptions for input validation.

```
Online_Banking_System.py U X

Online_Banking_System.py > LoanAccount > _init_
1    # Custom Exception for Invalid Inputs
2    class InvalidInputException(Exception):
3        pass
```

Figure: Exception Handling – by Hafij

2. Abstract Base Class to ensure implementation of core methods.

```
5    # Abstract Base Class (Demonstrates Abstraction)
6    from abc import ABC, abstractmethod
7
8    class Account(ABC):
9        def __init__(self, account_number, account_balance, name, phone, email):
10           if account_number <= 0:
11               raise InvalidInputException("Account number must be positive.")
12           if account_balance < 0:
13               raise InvalidInputException("Account balance cannot be negative.")
14           self.__account_number = account_number
15           self.__account_balance = account_balance
16           self.__name = name
17           self.__phone = phone
18           self.__email = email
19
20       @abstractmethod
21       def deposit(self, amount):
22           pass
23
24       @abstractmethod
25       def withdraw(self, amount):
26           pass
27
28       @abstractmethod
29       def display(self):
30           pass
```

Figure: Abstraction -by Hira

3.Multilevel Inheritance.

```python
70 ⌄ class SavingsAccount(Account):
71 ⌄     def __init__(self, account_number, account_balance, name, phone, email, interest_rate):
72           super().__init__(account_number, account_balance, name, phone, email)
73 >         if interest_rate < 0: ⋯
75           self.__interest_rate = interest_rate
76
77 >     def deposit(self, amount): ⋯
82
83 >     def withdraw(self, amount): ⋯
88
89 >     def display(self): ⋯
98
99 ⌄ class PremiumSavingsAccount(SavingsAccount):
100 ⌄     def __init__(self, account_number, account_balance, name, phone, email, interest_rate, rewards):
101           super().__init__(account_number, account_balance, name, phone, email, interest_rate)
102 ⌄         if rewards < 0:
103               raise InvalidInputException("Rewards cannot be negative.")
104           self.__rewards = rewards
105
106 ⌄     def redeem_rewards(self):
107           print(f"Redeeming {self.__rewards} rewards.")
108           self.__rewards = 0
109
110 ⌄     def display(self):
111           super().display()
112           print(f"Rewards: {self.__rewards}")
```

Figure: Inheritance – by Afjal

3.2   Performance Analysis

- Handles up to 1000 accounts without performance degradation.

- Exception handling ensures robustness against invalid inputs.

3.3   Results and Discussion

The system successfully demonstrates OOP principles like abstraction, inheritance, and polymorphism. Users can create, manage, and interact with different account types.

Chapter 4

Engineering Standards and Mapping

The "Engineering Standards and Mapping" chapter outlines the technical standards and best practices followed during the design and development of the Bank Management System. (Lutz, 2009)

4.1 Impact on Society, Environment and Sustainability

The Bank Management System positively impacts society, the environment, and long-term sustainability by addressing the following key areas:

4.1.1 Impact on Life

- The system improves efficiency in financial management by automating core operations such as deposits, withdrawals, and loan management.

- It reduces human errors in banking processes, ensuring greater accuracy and customer satisfaction.

- Enhances user convenience through simplified operations, saving time for both customers and staff.

4.1.2 Impact on Society & Environment

- Ensures data privacy and security by validating user inputs and preventing unauthorized access.

- Promotes fairness and transparency in banking operations, eliminating manual errors and biases.

- Aligns with ethical principles such as accountability and trustworthiness, which are essential for financial systems.

4.1.3 Ethical Aspects

- Ensures data privacy and security by validating user inputs and preventing unauthorized access.

- Promotes fairness and transparency in banking operations, eliminating manual errors and biases.

- Aligns with ethical principles such as accountability and trustworthiness, which are essential for financial systems.

4.1.4 Sustainability Plan

The system was designed with sustainability in mind by:

- Reducing paper-based processes, contributing to environmental conservation.

- Optimizing resource allocation through a lightweight, modular design, ensuring scalability and long-term usability.

- Offering a robust foundation for future enhancements like cloud-based solutions, which can minimize hardware dependencies.

4.2   Project Management and Team Work

The project was managed collaboratively, with clear roles and responsibilities assigned to each team member. The following approach was taken:

1.Task Distribution:
- Database Design: Designed relational tables and ensured normalization.
- System Development: Implemented Python classes, functions, and exception handling.
- Testing: Conducted rigorous testing to identify and resolve bugs.
- Documentation: Compiled project requirements, diagrams, and the final report.

2. Resource Utilization:
- Despite the absence of external funding, the team effectively utilized open-source tools and collaborative   communication platforms to streamline the process.

3. Progress Monitoring:
- Weekly progress meetings ensured the project stayed on track, with regular updates and issue resolutions.

4.3   Complex Engineering Problem

This section maps the complex engineering problems addressed in the project with Program Outcomes (POs) and engineering problem categories, ensuring a rationale is provided for each mapping.

4.3.1   Mapping of Program Outcome

The table below illustrates how the Bank Management System aligns with the targeted Program Outcomes (POs):

Table 4.1: Justification of Program Outcomes

PO's	Justification
PO1	Demonstrates a strong understanding of fundamental OOP concepts like abstraction, encapsulation, and inheritance while solving complex problems.
PO2	The design incorporates polymorphism, exception handling, and multiple inheritance to achieve optimized and reliable code solutions.
PO3	The project applies a systematic approach to problem-solving, including analysis, design, implementation, and testing of a complex system.

4.3.2   Complex Problem Solving

The Bank Management System addresses engineering problem categories defined by EP1–EP7. The table below maps these categories with their justification

Knowledge profile and rational thereof.

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	EP2 Range of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent Of Stakeholder Involvement	EP7 Inter-dependence
The project applies advanced OOP concepts like abstraction, polymorphism, and exception handling to solve real-world banking challenges.	Conflicting requirements such as performance optimization and ensuring modularity were balanced while developing the system.	The system was designed after thorough analysis of requirements, leading to robust features like account management	Issues such as handling invalid user inputs and ensuring reliable exception management	Standard programming practices and OOP principles were strictly followed to ensure a maintainable and scalable system.	The project considered both technical and non-technical users, ensuring the interface was simple and easy to navigate.	The system modules, such as account management, transactions, and loan management, were designed to work interdependently to achieve efficiency.

4.3.3 Engineering Activities

This section maps the project's engineering activities with categories EA1–EA5, providing rationale for each activity.

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
Utilized open-source tools like **Python** and **IDLE/VS Code**, ensuring cost-efficiency and accessibility to all team members.	Interaction with team members included collaborative discussions, code reviews, and systematic testing to refine system functionality.	Innovative use of OOP concepts like multiple inheritance, abstract base classes, and custom exceptions enhanced system robustness.	The project minimizes the need for manual, paper-based banking systems, thereby reducing resource waste and promoting sustainability.	Ensured familiarity with industry coding standards and OOP best practices, applying rigorous testing and validation procedures.

Chapter 5

Conclusion

The conclusion chapter will summarize the key achievements of the project, highlight its contributions to improving Banking management, and discuss potential future enhancements and applications.

5.1   Summary

This project successfully developed a Bank Management System utilizing Object-Oriented Programming (OOP) principles in Python. The system automates essential banking operations such as account creation, deposits, withdrawals, loan management, and notifications, providing a streamlined and modular solution for bank operations. By focusing on key OOP concepts like abstraction, inheritance, polymorphism, and encapsulation, the project demonstrates the ability to solve real-world problems with efficient design and implementation. **(Lutz, 2009)**

5.2   Limitation

Despite the project's achievements, several limitations were identified due to time constraints, resource availability, and the scope of the project:

Limited Time Frame:

- The project was completed within a 20-day timeline, which restricted the implementation of advanced features such as database integration, GUI development, and multi-user support.

- Extensive testing for edge cases and performance optimization was also constrained due to the tight schedule.

Console-Based Interface:

- The current system operates solely as a console-driven application, which, while functional, lacks a graphical user interface (GUI) to enhance user interaction and experience.

Absence of Persistent Storage:

- The project does not include persistent data storage mechanisms such as a database, limiting its ability to retain account or transaction data after program termination.

5.3   Future Work

Several enhancements can be incorporated into future versions of the Bank Management System to address the identified limitations and improve overall functionality:

Persistent Data Storage:

- Integrate a relational database such as MySQL or SQLite to enable persistent storage of

account information, transactions, and loan records.

- This will ensure data remains accessible even after the program is closed and can be retrieved for future operations.

Graphical User Interface (GUI):

- Develop a user-friendly front-end interface using technologies like Tkinter, PyQt, or frameworks such as HTML/CSS/JS for web applications.
- A GUI will enhance usability, making the system accessible to non-technical users and improving overall user experience.

Advanced Features:

- Add transaction history tracking, report generation, and loan interest calculations to extend the system's capabilities.
- Implement notification systems (e.g., SMS or email) to alert users about account activity or loan due dates.

Performance Optimization:

- Optimize the system to handle large-scale data and improve runtime efficiency, ensuring smooth operation for thousands of accounts and transactions.

Team Contribution:

1. Mehedi Hasan Hira   -   0242220005101742   -   Abstraction, Encapsulation, polymorphism
2. Md. Hafijur Rahman   -   0242220005101620   -   Exception Handling
3. Afjal   -   0242220005101730   -   Inheritance, Login option

- ➢ [GitHub Link](#)

References

Lutz, M. (2009). *Learning Python.* United States of America: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.