User Manual for Account Management System

Introduction

This document provides a simple guide for using the Account Management System. The program allows users to manage their accounts by performing actions such as depositing money, withdrawing funds, checking balances, and saving account details to a file for persistent storage. The program demonstrates key concepts of Object-Oriented Programming (OOP) and file handling in C++.

Functional Requirements

1. Account Creation

- Users must input their details, including:
  - Name
  - Identity Number
  - Email Address
  - Phone Number
  - KRA PIN
- An account is created using these details with an initial balance of 0.

2. Deposit Money

- Users can deposit money into their account.
- The system validates that the deposit amount is positive.

3. Withdraw Money

- Users can withdraw money from their account.
- The system ensures that the withdrawal amount does not exceed the available balance.

4. Check Balance

- Users can view their current account balance.
- The system prompts the user to perform a transaction after checking the balance.

5. Transaction Dashboard

- Users can:
  - Send money to a phone number.
  - Transfer funds to a bank account.

    - Send money to another account.
- The system validates that the transaction amount is within the available balance.

6. Save Account to File

- Account details are saved to a file (`account_data.txt`) for persistent storage.
- Details saved include:
    - Name
    - Identity Number
    - Email Address
    - Phone Number
    - KRA PIN
    - Account Balance

7. Load Account from File

- The system can load account details from `account_data.txt`.
- Users can view the loaded account's balance.

8. Exit

- Users can exit the program at any time via the menu.

Non-Functional Requirements

1. Usability

- The program uses a command-line interface (CLI) with simple menu options.
- Input prompts guide the user through each step.

2. Reliability

- Ensures data integrity by validating input values (e.g., positive deposit and withdrawal amounts).
- Handles insufficient balance scenarios gracefully.

3. Portability

- Designed to run on any system with a C++ compiler that supports standard libraries.

4. Performance

- File operations (read/write) are efficient and occur only when required.

5. Maintainability

- Modular design using OOP principles allows for easy extension and modification.
- Code is structured into distinct methods for each functionality.

6. Security

- Basic validation ensures no invalid or malicious input (e.g., negative amounts).
- User data is stored in plaintext; sensitive fields (like KRA PIN) can be encrypted in future updates.

How to Use

1. Compile and Run

1. Save the source code to a `.cpp` file.
2. Compile using a C++ compiler:

   ```
   g++ -o account_management account_management.cpp
   ```

3. Run the program:

   ```
   ./account_management
   ```

2. Navigate the Menu

- Follow on-screen prompts to select menu options (e.g., Deposit, Withdraw, Check Balance).
- Enter valid inputs when prompted.

3. Save and Load Account

- At the end of your session, account details are saved automatically to `account_data.txt`.
- The system can load and display previously saved account details on subsequent runs.

Future Enhancements

- Encrypt sensitive user data (e.g., KRA PIN) before saving to the file.
- Add graphical user interface (GUI) support.
- Include additional account types (e.g., Savings, Checking).
- Implement multi-user support.

End of Document

some of the difficulties we faced was trying to incorporate the oop principles especially polymorphism.