Bank Account System Documentation

Overview

This program simulates a simple bank account system using Object-Oriented Programming (OOP). It allows users to log in, perform basic banking operations, and manage their account data, which is stored in a JSON file. The program consists of two main components:

1. **`bank_account.py`**: Contains the `BankAccount` class to handle account operations.
2. **`main.py`**: The starting point of the application, handling user interactions.

Features

1. **Login System**:
    - Users must provide a valid username and password to access their account.
    - Invalid login attempts are prompted with an error message.
2. **Account Operations**:
    - **Check Balance**: Displays the users account balance and account type.
    - **Deposit**: Allows users to add money to their account. Input is formatted to two decimal places.
    - **Withdraw**: Allows users to withdraw money, and ensure they have sufficient balance. For “Savings Account”, only allow withdrawal if the remaining balance will be above $100.
    - **Logout**: Ends the session.
3. **Persistent Data**:
    - Account information is stored in a `JSON` file (`database.json`), including account_holder name, password, account type, and balance.
    - Changes made during a session (example: deposits or withdrawals) are saved to the file automatically.

Usage

Initial Setup

1. Create a `JSON` file named `database.json` to store account information in the following format:

banksystem > {} database.json > ...

```json
{
  "accounts": [
    {
      "account_holder": "user1",
      "account_type": "Savings",
      "balance": 5040.44,
      "password": "password1"
    },
    {
      "account_holder": "user2",
      "account_type": "Checking",
      "balance": 540.0,
      "password": "password2"
    }
  ]
}
```

2. Place both `bank_account.py` and `main.py` in the same directory as the `database.json` file.

Running the Program

1. Execute `main.py` either from a terminal or IDE, and ensure python is already installed.
2. Follow the prompts:
    - Enter your username and password to log in.
    - Select one of the options from the menu to perform an action.

Example Session

Start by running `main.py` on a terminal, and logging in with the given credentials:

```
┌──(kali㉿MSI)-[/mnt/c/Users/azizi/Desktop/assessment/banksystem]
└─$ python3 main.py
--------------------
Welcome to the Bank Account System!
Enter account holder name: user1
Enter your password: password1
Login successful!
--------------------

1. Check balance
2. Deposit
3. Withdraw
4. Logout
Choose an option >>
```

Check balance

```
1. Check balance
2. Deposit
3. Withdraw
4. Logout
Choose an option >> 1
--------------------
Balance: $4900.0
Type: Savings Account
--------------------
```

Deposit to account

```
1. Check balance
2. Deposit
3. Withdraw
4. Logout
Choose an option >> 2
--------------------
Enter amount to deposit: $177.88
Deposited $177.88. New balance: $5077.88
--------------------
```

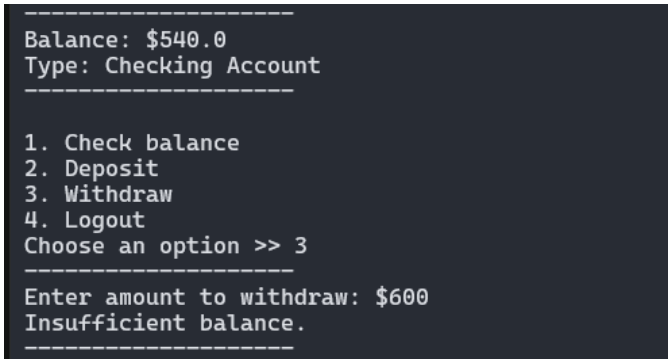
Withdraw from account

```
1. Check balance
2. Deposit
3. Withdraw
4. Logout
Choose an option >> 3
--------------------
Enter amount to withdraw: $37.44
Withdrew $37.44. New balance: $5040.44
--------------------
```

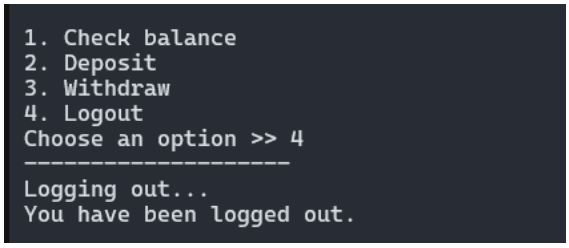
Invalid withdrawal (Savings Account type). It will not allow the balance to be below $100.

```
1. Check balance
2. Deposit
3. Withdraw
4. Logout
Choose an option >> 3
--------------------
Enter amount to withdraw: $99999
Insufficient balance. Your remaining balance should be $100 minimum.
--------------------
```

Invalid withdrawal (Checking Account type)



Log Out action. This will also exit the program



Limitations

1. The format of a JSON database structure could be easily destroyed by improper data handling. Not advisable for larger systems.
2. Passwords are stored in plain text in the JSON file. For production systems, encryption should be used for secure storage.
3. Input validation is basic and assumes proper user input for numerical values.

Future Improvements

1. **Using MySQL database**: It offers scalability, advanced querying, and better security, making it more suitable for handling this type of financial system.
2. **Password Encryption**: Use libraries like `bcrypt` or `hashlib` for secure password storage. Not stored as plaintext.
3. **Input Validation**: Add checks for invalid characters and stricter format enforcement.
4. **Advanced Features**:
    - Add transaction history logging.
    - Transfer functionality between accounts.

- Expand the features for multiple account types (savings, checking).
- Implement account registration.

In conclusion, this program represents what I was able to develop within the limited time available. While it includes core functionality like account management and basic banking operations, I recognise that there is room for improvements and additional features that could be implemented with more time and planning.