Building a Console-Based ATM System in Python

I recently worked on an interesting project to create a console-based ATM system using Python. This project involves multiple classes to handle different functionalities, simulating a real-world ATM. Below is an overview of the structure and the detailed implementation.

Class Structure and Responsibilities

1. **User**: This class represents a bank customer. It includes methods for verifying the user's PIN to ensure secure access to the account.
2. **Account**: The Account class manages the user's bank account. It handles the account balance and maintains a transaction history. Methods include depositing and withdrawing money, and transferring funds to another account.
3. **Transaction**: Each transaction (withdrawal, deposit, or transfer) is recorded as an instance of the Transaction class. This class captures the type of transaction and the amount involved.
4. **Bank**: This class simulates the bank itself. It contains a collection of users and provides methods to add new users and retrieve existing users based on their user ID.
5. **ATM**: The ATM class is the heart of the system. It manages user interactions, including authentication and various banking operations. It presents a menu to the user after successful login, allowing access to transaction history, withdrawals, deposits, and transfers.

Implementation Details

Here's a breakdown of how each part of the system works:

User Class

The `User` class holds the user's ID, PIN, and name. It also contains an instance of the `Account` class, representing the user's bank account. The method `verify pin` checks if the entered PIN matches the user's PIN.

CODE:

```
class User:

    def __init__(self, user_id, pin, name):

        self.user_id = user_id

        self.pin = pin

        self.name = name
        self.account = Account()


    def verify_pin(self, pin):

        return self.pin == pin
```

Account Class

The `Account` class manages the account balance and the transaction history. It includes methods to deposit and withdraw money, as well as to transfer money to another account. The `get_transaction_history` method returns a list of transactions.

CODE:

```
class Account:

    def __init__(self):

        self.balance = 0

        self.transaction_history = []


    def deposit(self, amount):

        self.balance += amount

        self.transaction_history.append(Transaction("Deposit", amount))


    def withdraw(self, amount):

        if amount > self.balance:

            return False

        self.balance -= amount

        self.transaction_history.append(Transaction("Withdraw", amount))

        return True


    def transfer(self, amount, target_account):
        if amount > self.balance:
            return False
        self.balance -= amount
        target_account.balance += amount
        self.transaction_history.append(Transaction("Transfer", amount))
        return True

    def get_transaction_history(self):
        return self.transaction_history
```

Transaction Class

The `Transaction` class records each transaction type and amount.

CODE:

```
class Transaction:
    def __init__(self, transaction_type, amount):
        self.transaction_type = transaction_type
        self.amount = amount
```

Bank Class

The `Bank` class holds all the users. It provides methods to add new users and retrieve users by their user ID.

CODE:

```
class Bank:
    def __init__(self):
        self.users = {}

    def add_user(self, user):
        self.users[user.user_id] = user


    def get_user(self, user_id):
        return self.users.get(user_id, None)
```

ATM Class

The `ATM` class is responsible for interacting with the user. It handles user authentication and displays the menu for different operations once the user is authenticated. It includes methods for each operation available in the menu.

CODE:

```
class ATM:
    def __init__(self, bank):
        self.bank = bank
        self.current_user = None


    def authenticate_user(self):
        user_id = input("Enter user ID: ")
        pin = input("Enter PIN: ")
        user = self.bank.get_user(user_id)
        if user and user.verify_pin(pin):
            self.current_user = user
            print(f"Welcome {user.name}!")
            return True
        else:
            print("Invalid user ID or PIN.")
            return False
```

```
def show_menu(self):
    while True:
        print("\nATM Menu:")
        print("1. Transactions History")
        print("2. Withdraw")
        print("3. Deposit")
        print("4. Transfer")
        print("5. Quit")
        choice = input("Choose an option: ")

        if choice == '1':
            self.show_transaction_history()
        elif choice == '2':
            self.withdraw()
        elif choice == '3':
            self.deposit()
        elif choice == '4':
            self.transfer()
        elif choice == '5':
            print("Thank you for using the ATM. Visit Again!")
            sys.exit()
        else:
            print("Invalid choice. Please try again.")

def show_transaction_history(self):
    transactions = self.current_user.account.get_transaction_history()
    if not transactions:
        print("No transactions available.")
        return
    for transaction in transactions:
        print(f"{transaction.transaction_type}: ${transaction.amount}")

def withdraw(self):
    amount = float(input("Enter amount to withdraw: "))
    if self.current_user.account.withdraw(amount):
        print(f"${amount} withdrawn successfully.")
    else:
        print("Insufficient balance.")

def deposit(self):
    amount = float(input("Enter amount to deposit: "))
    self.current_user.account.deposit(amount)
    print(f"${amount} deposited successfully.")

def transfer(self):
    target_user_id = input("Enter the user ID to transfer to: ")
    target_user = self.bank.get_user(target_user_id)
    if not target_user:
        print("Invalid user ID.")
        return
    amount = float(input("Enter amount to transfer: "))

    if self.current_user.account.transfer(amount, target_user.account):

        print(f"${amount} transferred successfully to {target_user.name}.")

    else:

        print("Insufficient balance.")
```

Putting It All Together

Finally, we initialize the `Bank` and `ATM` classes and simulate adding users to the bank. The `ATM` instance handles user authentication and presents the menu for further operations.

CODE:

```
if __name__ == "__main__":

    # Sample data

    bank = Bank()

    bank.add_user(User("anil", "1234", "Anil"))

    bank.add_user(User("kumar", "5678", "Kumar"))



    atm = ATM(bank)

    if atm.authenticate_user():

        atm.show_menu()
```