Digital Wallet

In this project, you will create your first object-oriented financial application. It focuses on managing attributes and methods to simulate a real-world digital wallet system.

Module: `DigitalWallet`

- **Attributes:**
  - `balance`: Stores the current wallet balance.
  - `transaction_history`: Keeps a log of all transactions.
- **Operations:**
  - `initialize_wallet()`
    - Sets up the wallet with an initial balance of 0 and empty transaction history.
  - `display_balance()`
    - Shows the current balance to the user.
  - `add_funds(amount)`
    - Adds funds to the wallet and records the transaction.
    - Validates that the amount is positive.
  - `make_payment(amount)`
    - Deducts funds if the balance is sufficient and logs the transaction.
  - `view_transaction_history()`
    - Displays all transaction records (credits and debits).
  - `run_wallet_interface()`
    - Provides a menu-driven interface for wallet operations.

Test Case Format

To validate the implementation, test cases will specify the method name, attributes, and inputs in a structured format. Your program should:

1. Read the test case inputs to create objects of `DigitalWallet`.
2. Call the specified methods with the given inputs.
3. Compare the outputs with the expected results to ensure correctness.

Example Test Case:

Method: initialize_wallet
Attributes: None
Expected Output: Wallet initialized with balance 0 and empty transaction history.

Method: add_funds
Attributes: balance=0, transaction_history=[]

Inputs: amount=100
Expected Output: Balance updated to 100, transaction history logged.

Method: make_payment
Attributes: balance=100, transaction_history=["+100"]
Inputs: amount=50
Expected Output: Balance updated to 50, transaction history logged.

Method: view_transaction_history
Attributes: balance=50, transaction_history=["+100", "-50"]
Expected Output: ["+100", "-50"]

Write your code to process test cases programmatically, ensuring the methods and attributes align with the specified inputs and expected results.

In this project, you will create your first object-oriented financial application. It focuses on managing attributes and methods to simulate a real-world digital wallet system.

Module: `DigitalWallet`

- **Attributes:**
  - `balance`: Stores the current wallet balance.
  - `transaction_history`: Keeps a log of all transactions.
- **Operations:**
  - `initialize_wallet()`
    - Sets up the wallet with an initial balance of 0 and empty transaction history.
  - `display_balance()`
    - Shows the current balance to the user.
  - `add_funds(amount)`
    - Adds funds to the wallet and records the transaction.
    - Validates that the amount is positive.
  - `make_payment(amount)`
    - Deducts funds if the balance is sufficient and logs the transaction.
  - `view_transaction_history()`
    - Displays all transaction records (credits and debits).
  - `run_wallet_interface()`
    - Provides a menu-driven interface for wallet operations.

Instructions for Implementation

1. Use the given module descriptions to design your program in any object-oriented programming language.
2. Ensure that each operation is implemented as described, adhering to core OOP principles.
3. Test each functionality independently before integrating into a complete application.
4. Enhance the implementation with input validation and user-friendly feedback.
