Flipkart Mini Wallet

Problem Statement

Design a prepaid mini wallet for Flipkart through which users can send and receive money among themselves and can get cashback in wallet. The user can load money into the wallet via various modes of sources like Credit Card, Debit Card or UPI etc and perform easy transactions.

Time Window: 90 Minutes

Requirements

- Users need to register on Flipkart to use this wallet.
- The user can load money into his wallet via various sources (Credit Card, Debit Card, UPI etc).
  - Minimum amount of load money should be greater than 0.
  - Assumption - No need for Integration from sources, can be implemented just success acknowledged.
- The user can send money to other users from their wallet.
  - The minimum amount of transaction will be always greater the 0
  - The user must have a sufficient balance in his wallet while doing the transaction.
- The user can fetch their wallet balance at any point of time, this should consider both credit & debit type of transactions.
- The user can get transaction history based on
  - Sort their all transactions history
    - Based on Amount (“amount”)
    - Based on Transaction DateTime (“time”)
  - Filter transactions history
    - Based on Send or Receive Amount (“send/receive”)
  - Assumption: Consider single filter & sorted field

BONUS

- The transactions will be eligible for cashback if they meet their respective criteria. Let's say criteria can be, based on “after each 5 transactions” or “With 0.5 probability for each transaction” ', provide X amount as cashback and added to the user's wallet.

Capabilities
Below are various functions that should be supported with the necessary parameters passed.
These are just suggestions, the signatures can be altered as long as the functionalities are provided.

- registerUser() - Register user to use wallet
- topUpWallet() - Add money to the wallet for users with any of the resources.
- fetchBalance() - Fetch balance available in the wallet for user
- sendMoney() - Send money to the user. Each money transfer will be considered as a transaction.
- getTransactions(filter, sorter) - fetches transactions history based on filter and sorting criteria
- getCashback() - Get total cashback amount for eligible transactions.

Testcase

INPUT:

```
fetchBalance("Bob")

registerUser("Bob")
topUpWallet("Bob", "CC", 1000)
topUpWallet("Bob", "UPI", 100)
fetchBalance("Bob")

registerUser("Alice")
topUpWallet("Alice", "CC", 100)
fetchBalance("Alice")

sendMoney("Bob", "Alice", 1250)
sendMoney("Bob", "Alice", 250)
sendMoney("Alice", "Bob", 50)

fetchBalance("Bob")
fetchBalance("Alice")

getTransactions("Bob", "send", "amount")
getTransactions("Bob", "receive", "time")

BONUS:
=> Let's say On the below transaction, Bob got a callback of 10 Rs based on
criteria defined.

sendMoney("Bob", "Alice", 100)
getCashbask("Bob")

fetchBalance("Bob")
fetchBalance("Alice")

getTransactions("Bob", "send", "amount")
getTransactions("Bob", "receive", "time")
```

OUTPUT:

```
"User Bob is not registered"

"User Bob is registered"
"Bob's wallet has credit with 1000 Rs successfully"
"Bob's wallet has credit with 100 Rs successfully"
"Bob’s wallet has 1100 Rs amount"

"User Alice is registered"
"Alice’s wallet has credit with 100 Rs successfully"
"Alice’s wallet has 100 Rs amount"

"Bob doesn’t have sufficient amount to transfer 1250 Rs"
"Bob has transferred 250 Rs amount to Alice"
"Alice has transferred 50 Rs amount to Bob"

"Bob’s wallet has 900 Rs amount"
"Alice’s wallet has 300 Rs amount"

Bob -> Alice : 250 Rs
Bob -> Alice : 1250 Rs

Alice -> Bob : 50 Rs

BONUS OUTPUT:
"Bob has transferred 100 Rs amount to Alice and get cashback of 10 Rs"
"Bob got total cashback of 10 Rs"

"Bob’s wallet has 800 Rs amount"
"Alice’s wallet has 400 Rs amount"

Bob -> Alice : 100 Rs
Bob -> Alice : 250 Rs
Bob -> Alice : 1250 Rs

Alice -> Bob : 50 Rs
```

Expectations:

1. Create the sample data yourself. You can put it into a file, test case or main driver program or via Unit tests itself.
2. Database integration is not required. You should store the data in memory using a language-specific data structure.
3. The code should be demo-able. Either by using the main driver program or test cases.
4. The code should be modular. The code should have the basic OO design. Please do not jam in the responsibilities of one class into another.
5. The code should be extensible. Wherever applicable, use interfaces and contracts between different methods. It should be easy to add/remove functionality without rewriting the entire codebase.
6. The code should handle edge cases properly and fail gracefully.
7. The code should be legible, readable and DRY.

Guidelines:

1. Please do not access the internet for anything EXCEPT syntax.
2. You are free to use the language and IDE of your choice.
3. The entire code should be your own.
4. Time Window : 90 Minutes