Machine Coding - Expense Sharing

Problem Statement

Create an expense sharing application.
An expense sharing application is where you can add your expenses and split it among different people. The app keeps balances between people as in who owes how much to whom.

Example
You live with 3 other friends.
You: User1 (id: u1)
Flatmates: User2 (u2), User3 (u3), User4 (u4)

This month's electricity bill was Rs. 1000.
Now you can just go to the app and add that you paid 1000, select all 4 people and then select split equally.
Input: **u1 1000 4 u1 u2 u3 u4 EQUAL**

For this transaction, everyone owes 250 to User1.
The app should update the balances in each of the profiles accordingly.

User2 owes User1: 250 (0+250)
User3 owes User1: 250 (0+250)
User4 owes User1: 250 (0+250)

Now, you order food from SWIGGY, for User2 and User3 as they asked you to.
The total amount for each person is different.
Input: **u1 1250 2 u2 u3 EXACT 370 880**

For this transaction, User2 owes 370 to User1 and User3 owes 880 to User1.

The app should update the balances in each of the profiles accordingly.
User2 owes User1: 620 (250+370)
User3 owes User1: 1130 (250+880)
User4 owes User1: 250 (250+0)

Requirements

User: Each user should have a userId, name, email, mobile number.
Expense: Could either be EQUAL, EXACT

Users can add any amount, select any type of expense, and split with any of the available users.
In the case of exact, you need to verify if the total sum of shares is equal to the total amount or not.
The application should have the capability to show expenses for a single user as well as balances for everyone.
When asked to show balances, the application should show the balances of a user with all the users where there is a non-zero balance.
The amount should be rounded off to two decimal places. Say if User1 paid 100 and amount is split equally among 3 people. **Assign 33.34 to first-person and 33.33 to others.**

Input

You can create a few users in your main method. No need to take it as input.
There will be 3 types of input:

Expense in the format: EXPENSE <user-id-of-person-who-paid> <no-of-users> <space-separated-list-of-users> <EQUAL/EXACT> <space-separated-values-in-case-of-non-equal>
Show balances for all: SHOW
Show balances for a single user: SHOW <user-id>

Output

When asked to show balance for a single user. Show all the balances that user is part of:
Format: <user-id-of-x> owes <user-id-of-y>: <amount>
If there are no balances for the input, print No balances
In cases where the user for which balance was asked for, owes money, they'll be x. They'll be y otherwise.

Input	Output
SHOW	No balances
SHOW user1	No balances
EXPENSE user1 1000 4 user1 user2 user3 user4 EQUAL	OK
SHOW user4	User4 owes User1: 250
SHOW user1	User2 owes User1: 250 User3 owes User1: 250 User4 owes User1: 250

EXPENSE user2 1000 2 user1 user4 EXACT 250 750	OK
SHOW user2	User4 owes User2: 750
SHOW user1	User3 owes User1: 250 User4 owes User1: 250
SHOW	User3 owes User1: 250 User4 owes User2: 750 User4 owes User1: 250

Expectations

Clean and object-oriented low-level design.
Appropriate coding conventions and directory structure wrt language used.
Error handling of edge cases.
Exception handling.
Optimality of solution in terms of time complexity for various operations

Note:

1. You need not focus more on how to take input. Please feel free to take input in any form you like(sysin, test cases, test file, test functions, static inits).
2. No need to create a GUI or use database.