APIs:

1. AddExpense(User paid, Integer number, List<> Users, Action action, List<> Expenses, Group group)
2. ShowExpense(User paid)
3. ShowAllExpense( )
4. CreateGroup(User owner, List<> Users)

```
Enum Action {
        Exact,
        Equal,
        Percentage
}
```

```
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;

public class TransactionManager {
    List<User> users;
    List<Transaction> transactions;
    Map<User,Map<User, Double>> userMapping;

    ReadWriteLock rwLock = new ReadWriteLock() {
        @Override
        public Lock readLock() {
            return null;
        }

        @Override
        public Lock writeLock() {
            return null;
        }
    }
    TransactionManager () {
        users = new ArrayList<>();
        transactions = new ArrayList<>();
        userMapping = new ConcurrentHashMap<>();
    }
    public void addUser(User user) {
        this.users.add(user);

    }

    public void AddExpense(User paid, Integer nOfUsers, List<User> user, String action,
                          Double amount, List<Double> expenses ) {

        Action act = new Action();
        Transaction transaction = act.addExpenseFactory(paid,
nOfUsers,user,action,amount,expenses);
        transactions.add(transaction);

        this.rwLock.writeLock().lock();
        Map<User, Double> owner;
        if(!userMapping.containsKey(paid)) {
            owner = new ConcurrentHashMap<>();
            for (int i = 0; i < nOfUsers; i++) {
                owner.put(transaction.userList.get(i), transaction.expenses.get(i));
            }
        } else {
            owner = this.userMapping.get(paid);
            for (int i = 0; i < nOfUsers; i++) {
                User owe = transaction.userList.get(i);
                if(owner.containsKey(owe)) {
                    owner.put(owe, owner.get(owe) + transaction.expenses.get(i));
                } else {
                    owner.put(owe, transaction.expenses.get(i));
                }
            }
        }
        userMapping.put(paid, owner);
        this.rwLock.writeLock().unlock();

    }
    public  void ShowExpense( User paid ) {
        if(userMapping.containsKey(paid)) {

            Map<User, Double> owner = userMapping.get(paid);
            for(Map.Entry<User, Double> entry : owner.entrySet()) {
                System.out.println(entry.getKey().name + " owes user " + paid.name + " :" +
entry.getValue());
            }
        }
        System.out.println("User does not exist");
    }
    public  void ShowAllExpense( ) {

    }
    public  void  CreateGroup( User owner, List<> Users ) {

    }

}



import java.util.*;

public class Transaction {
    Integer transactionId;
    User paid;
    Integer nOfUsers;
    List<User> userList;
    Double amount;
    List<Double> expenses;

    Integer generateRandomId() {
        return 1;
    }
    Transaction(User paid, Integer nOfUsers, List<User> user, Double amount, List<Double>
expenses) {
        this.transactionId = generateRandomId();
        this.paid = paid;
        this.nOfUsers = nOfUsers;
        this.userList = user;
        this.amount = amount;
        this.expenses = expenses;
    }
}


import java.time.Clock;
import java.util.*;

public class Action {
    String[] actions = new String[]{"Exact", "Equal", "Percentage"};

    public boolean validateExactSplit(List<Double> expenses, Double amount) {
        double sum = 0.0;
        for(int i = 0; i < expenses.size();i++) {
            sum += expenses.size();
        }
        return sum == amount;
    }

    public boolean validatePercentSplit(List<Double> expenses, Double amount) {
        double sum = 0.0;
        for(int i = 0; i < expenses.size();i++) {
            sum += expenses.size();
        }
        return sum == 100;
    }

    public Transaction addExpenseFactory(User paid, Integer nOfUsers, List<User> user, String
action,
                                    Double amount, List<Double> expenses) {
        if(action == this.actions[0]) {
            if(this.validateExactSplit(expenses, amount)) {
                return new Transaction(paid, nOfUsers, user, amount, expenses);
            }
        } else if(action == this.actions[1]) {
            List<Double> expense = new ArrayList<>();
            for(int i = 0; i < nOfUsers; i++) {
                expense.add(amount/nOfUsers);
            }
            return new Transaction(paid, nOfUsers, user, amount, expense);

        } else if(action == this.actions[2]) {
            if(this.validatePercentSplit(expenses, amount)) {
                List<Double> expense = new ArrayList<>();
                for(int i = 0; i < nOfUsers; i++) {
                    expense.add(expenses.get(i) * amount / 100);
                }
                return new Transaction(paid, nOfUsers, user, amount, expenses);
            }

        } else {
            System.out.println("Invalid Action");
        }
        return null;
    }


}

public class User {

    Integer id;
    String name;
    String email;
    String phoneNo;

    public User(Integer id, String name, String email, String phone ) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.phoneNo = phone;
    }


}
```