35. Splitwise

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 the 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, It is the BBD sale on Flipkart and there is an offer on your card.

You buy a few stuffs 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)

Now, you go out with your flatmates and take your brother/sister along with you.

User4 pays and everyone splits equally. You owe for 2 people.

Input: u4 1200 4 u1 u2 u3 u4 PERCENT 40 20 20 20

For this transaction, User1 owes 480 to User4, User2 owes 240 to User4 and User3 owes 240 to User4.

The app should update the balances in each of the profiles accordingly.

- User1 owes User4: 230 (250-480)
- User2 owes User1: 620 (620+0)
- User2 owes User4: 240 (0+240)
- User3 owes User1: 1130 (1130+0)
- User3 owes User4: 240 (0+240)

Requirements

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

- Users can add any amount, select any type of expense and split with any of the available users.
- The percent and amount provided could have decimals upto two decimal places.

In case of percent, you need to verify if the total sum of percentage shares is 100 or not.

In 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 a capability to show expenses for a single user as well as balances for everyone.

When asked to show balances, the application should show 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/PERCENT> <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.

Optional Requirements

- A way to add an expense name while adding the expense. Can also add notes, images, etc.
- Option to split by share. Ex: 'User4 pays and everyone splits equally. You pay for 2 people.' could be added as: u4 1200 4 u1 u2 u3 u4 SHARE 2 1 1 1
- A way to show the passbook for a user. The entries should show all the transactions a user was part of. You can print in any format you like.
- There can be an option to simplify expenses. When simplify expenses is turned on (is true), the balances should get simplified. Ex: 'User1 owes 250 to User2 and User2 owes 200 to User3' should simplify to 'User1 owes 50 to User2 and 200 to User3'.

There could be different ways to design a solution for this. I'll mention how I would have approached it during an actual interview.

Let’s dissect the problem statement to determine how to design a good solution for it.

The gist of the problem statement is that we need to create an expense sharing application. The application will have multiple users, an option to add expenses (EQUAL, EXACT or PERCENT) and an option to show balances (All or a specific user).

Here I’ve decided that I’ll keep a Driver class to take input from the user and a ExpenseManager class to orchestrate the application.

In the interview, I’ll not focus on solving with the optional requirements but will try to keep my design such that if I’ve time, I can do minimal changes to add them. Let’s look at each of those requirements one-by-one and see which ones I would want to ignore to keep my design clean. The problem is supposed to be solved in 2 hours and so I do not want to get stuck in something that is time-consuming but optional.

For optional requirement #1, I am anyway keeping an Expense model class so adding attributes should be easy.

For optional requirement #2, to keep my code extensible, I’ll anyway be having different types of splits so adding another should be easy.

For optional requirement #3, I’ll keep an expense list in the ExpenseManager.

For optional requirement #4, I’ll have to take extra care of not messing up the logic and so ignoring this optional requirement for now. Let’s see if my design can accommodate that without much change.

So, I can keep my code extensible for 3 of the 4 optional requirements!

Then, I'll note down the entities (models) that will be involved in the design.

User

- Split (EqualSplit, ExactSplit, PercentSplit)
- ExpenseMetadata (For Optional requirement #1)
- Expense (EqualExpense, ExactExpense, PercentExpense)
- ExpenseType (enum to differentiate between different expense types)

Then, I'll create the models for all of these. Keeping different sub-classes here for 'Split' to abstract the logic of how the split amounts are calculated from the caller and also to keep the actual values.

User.java

```
package com.workattech.splitwise.models;

public class User {
	private String id;
	private String name;
	private String email;
	private String phone;

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

	public String getId() {
		return id;
	}
```

```
    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

Split.java

```
package com.workattech.splitwise.models.split;

import com.workattech.splitwise.models.User;
```

```
public abstract class Split {
	private User user;
	double amount;

	public Split(User user) {
		this.user = user;
	}

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}

	public double getAmount() {
		return amount;
	}

	public void setAmount(double amount) {
		this.amount = amount;
	}
}
```

EqualSplit.java

```
package com.workattech.splitwise.models.split;

import com.workattech.splitwise.models.User;

public class EqualSplit extends Split {
```

```
    public EqualSplit(User user) {
        super(user);
    }
}
```

ExactSplit.java

```
package com.workattech.splitwise.models.split;

import com.workattech.splitwise.models.User;

public class ExactSplit extends Split {

    public ExactSplit(User user, double amount) {
        super(user);
        this.amount = amount;
    }
}
```

PercentSplit.java

```
package com.workattech.splitwise.models.split;

import com.workattech.splitwise.models.User;

public class PercentSplit extends Split {
    double percent;

    public PercentSplit(User user, double percent) {
        super(user);
        this.percent = percent;
    }
```

```
    public double getPercent() {
        return percent;
    }

    public void setPercent(double percent) {
        this.percent = percent;
    }
}
```

ExpenseMetadata.java

```
package com.workattech.splitwise.models.expense;

public class ExpenseMetadata {
    private String name;
    private String imgUrl;
    private String notes;

    public ExpenseMetadata(String name, String imgUrl, String notes) {
        this.name = name;
        this.imgUrl = imgUrl;
        this.notes = notes;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getImgUrl() {
        return imgUrl;
```

```
    }

    public void setImgUrl(String imgUrl) {
        this.imgUrl = imgUrl;
    }

    public String getNotes() {
        return notes;
    }

    public void setNotes(String notes) {
        this.notes = notes;
    }
}
```

Expense.java

```
package com.workattech.splitwise.models.expense;

import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.split.Split;

import java.util.List;

public abstract class Expense {
    private String id;
    private double amount;
    private User paidBy;
    private List<Split> splits;
    private ExpenseMetadata metadata;

    public Expense(double amount, User paidBy, List<Split> splits, ExpenseMetadata metadata) {
        this.amount = amount;
        this.paidBy = paidBy;
```

```
        this.splits = splits;
        this.metadata = metadata;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public User getPaidBy() {
        return paidBy;
    }

    public void setPaidBy(User paidBy) {
        this.paidBy = paidBy;
    }

    public List<Split> getSplits() {
        return splits;
    }

    public void setSplits(List<Split> splits) {
        this.splits = splits;
    }
```

```
    public ExpenseMetadata getMetadata() {
        return metadata;
    }

    public void setMetadata(ExpenseMetadata metadata) {
        this.metadata = metadata;
    }

    public abstract boolean validate();
}
```

EqualExpense.java

```
package com.workattech.splitwise.models.expense;

import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.split.EqualSplit;
import com.workattech.splitwise.models.split.Split;

import java.util.List;

public class EqualExpense extends Expense {
    public EqualExpense(double amount, User paidBy, List<Split> splits, ExpenseMetadata
expenseMetadata) {
        super(amount, paidBy, splits, expenseMetadata);
    }

    @Override
    public boolean validate() {
        for (Split split : getSplits()) {
            if (!(split instanceof EqualSplit)) {
                return false;
            }
        }
```

```
            return true;
        }
}
```

ExactExpense.java

```
package com.workattech.splitwise.models.expense;

import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.split.ExactSplit;
import com.workattech.splitwise.models.split.Split;

import java.util.List;

public class ExactExpense extends Expense {
    public ExactExpense(double amount, User paidBy, List<Split> splits, ExpenseMetadata expenseMetadata) {
        super(amount, paidBy, splits, expenseMetadata);
    }

    @Override
    public boolean validate() {
        for (Split split : getSplits()) {
            if (!(split instanceof ExactSplit)) {
                return false;
            }
        }

        double totalAmount = getAmount();
        double sumSplitAmount = 0;
        for (Split split : getSplits()) {
            ExactSplit exactSplit = (ExactSplit) split;
            sumSplitAmount += exactSplit.getAmount();
        }
```

```
            if (totalAmount != sumSplitAmount) {
                return false;
            }

            return true;
        }
}
```

PercentExpense.java

```
package com.workattech.splitwise.models.expense;

import com.workattech.splitwise.models.User;

import com.workattech.splitwise.models.split.PercentSplit;

import com.workattech.splitwise.models.split.Split;

import java.util.List;

public class PercentExpense extends Expense {

    public PercentExpense(double amount, User paidBy, List<Split> splits, ExpenseMetadata
expenseMetadata) {

        super(amount, paidBy, splits, expenseMetadata);

    }

    @Override

    public boolean validate() {

        for (Split split : getSplits()) {

            if (!(split instanceof PercentSplit)) {

                return false;

            }

        }
```

```
            double totalPercent = 100;
            double sumSplitPercent = 0;
            for (Split split : getSplits()) {
                PercentSplit exactSplit = (PercentSplit) split;
                sumSplitPercent += exactSplit.getPercent();
            }
            if (totalPercent != sumSplitPercent) {
                return false;
            }
            return true;
        }
}
```

ExpenseType.java

```
package com.workattech.splitwise.models.expense;

public enum ExpenseType {
    EQUAL,
    EXACT,
    PERCENT
}
```

Now that I have created the models, let me create the services which will be used to talk to the models and to each other.

Here, I'm creating an ExpenseService which will be used to talk to 'Expense'. As of now, it is only being used to create an Expense object depending on the ExpenseType.

ExpenseService.java

```
package com.workattech.splitwise;

import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.expense.*;
import com.workattech.splitwise.models.split.PercentSplit;
import com.workattech.splitwise.models.split.Split;

import java.util.List;

public class ExpenseService {
    public static Expense createExpense(ExpenseType expenseType, double amount, User paidBy,
List<Split> splits, ExpenseMetadata expenseMetadata) {
        switch (expenseType) {
            case EXACT:
                return new ExactExpense(amount, paidBy, splits, expenseMetadata);
            case PERCENT:
                for (Split split : splits) {
                    PercentSplit percentSplit = (PercentSplit) split;
                    split.setAmount((amount*percentSplit.getPercent())/100.0);
                }
                return new PercentExpense(amount, paidBy, splits, expenseMetadata);
            case EQUAL:
                int totalSplits = splits.size();
                double splitAmount = ((double) Math.round(amount*100/totalSplits))/100.0;
                for (Split split : splits) {
                    split.setAmount(splitAmount);
                }
                splits.get(0).setAmount(splitAmount + (amount - splitAmount*totalSplits));
                return new EqualExpense(amount, paidBy, splits, expenseMetadata);
            default:
```

```
                return null;
            }
        }
}
```

Now, I'll create the orchestrator 'ExpenseManager' which the 'Driver' class will talk to. ExpenseManager will store the user and expense data.

ExpenseManager.java

```
package com.workattech.splitwise;

import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.expense.Expense;
import com.workattech.splitwise.models.expense.ExpenseMetadata;
import com.workattech.splitwise.models.expense.ExpenseType;
import com.workattech.splitwise.models.split.Split;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ExpenseManager {
    List<Expense> expenses;
    Map<String, User> userMap;
    Map<String, Map<String, Double> > balanceSheet;

    public ExpenseManager() {
        expenses = new ArrayList<Expense>();
        userMap = new HashMap<String, User>();
        balanceSheet = new HashMap<String, Map<String, Double>>();
    }
```

```
    public void addUser(User user) {
        userMap.put(user.getId(), user);
        balanceSheet.put(user.getId(), new HashMap<String, Double>());
    }

    public void addExpense(ExpenseType expenseType, double amount, String paidBy, List<Split>
splits, ExpenseMetadata expenseMetadata) {
        Expense expense = ExpenseService.createExpense(expenseType, amount,
userMap.get(paidBy), splits, expenseMetadata);
        expenses.add(expense);
        for (Split split : expense.getSplits()) {
            String paidTo = split.getUser().getId();
            Map<String, Double> balances = balanceSheet.get(paidBy);
            if (!balances.containsKey(paidTo)) {
                balances.put(paidTo, 0.0);
            }
            balances.put(paidTo, balances.get(paidTo) + split.getAmount());

            balances = balanceSheet.get(paidTo);
            if (!balances.containsKey(paidBy)) {
                balances.put(paidBy, 0.0);
            }
            balances.put(paidBy, balances.get(paidBy) - split.getAmount());
        }
    }

    public void showBalance(String userId) {
        boolean isEmpty = true;
        for (Map.Entry<String, Double> userBalance : balanceSheet.get(userId).entrySet()) {
            if (userBalance.getValue() != 0) {
                isEmpty = false;
                printBalance(userId, userBalance.getKey(), userBalance.getValue());
            }
        }
```

```
            if (isEmpty) {
                System.out.println("No balances");
            }
        }

        public void showBalances() {
            boolean isEmpty = true;
            for (Map.Entry<String, Map<String, Double>> allBalances : balanceSheet.entrySet()) {
                for (Map.Entry<String, Double> userBalance : allBalances.getValue().entrySet()) {
                    if (userBalance.getValue() > 0) {
                        isEmpty = false;
                        printBalance(allBalances.getKey(), userBalance.getKey(),
userBalance.getValue());
                    }
                }
            }

            if (isEmpty) {
                System.out.println("No balances");
            }
        }

        private void printBalance(String user1, String user2, double amount) {
            String user1Name = userMap.get(user1).getName();
            String user2Name = userMap.get(user2).getName();
            if (amount < 0) {
                System.out.println(user1Name + " owes " + user2Name + ": " + Math.abs(amount));
            } else if (amount > 0) {
                System.out.println(user2Name + " owes " + user1Name + ": " + Math.abs(amount));
            }
        }
}
```

Note: I could have used enums instead of different Expense classes for a simpler design but avoided as I wanted to keep the solution language agnostic. If you're solving this in Java, you may try solving it using enums. I could also have used interfaces at certain places but I avoided it here for simplicity.

Now, that our core design is done, let's create the Driver class where we would take user input to add expenses to the ExpenseManager and to show balances as well.

Driver.java

```
package com.workattech.splitwise;
import com.workattech.splitwise.models.User;
import com.workattech.splitwise.models.expense.ExpenseType;
import com.workattech.splitwise.models.split.EqualSplit;
import com.workattech.splitwise.models.split.ExactSplit;
import com.workattech.splitwise.models.split.PercentSplit;
import com.workattech.splitwise.models.split.Split;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Driver {
	public static void main(String[] args) {
		ExpenseManager expenseManager = new ExpenseManager();


		expenseManager.addUser(new User("u1", "User1", "gaurav@workat.tech",
"9876543210"));
		expenseManager.addUser(new User("u2", "User2", "sagar@workat.tech",
"9876543210"));
		expenseManager.addUser(new User("u3", "User3", "hi@workat.tech", "9876543210"));
		expenseManager.addUser(new User("u4", "User4", "mock-interviews@workat.tech",
"9876543210"));


		Scanner scanner = new Scanner(System.in);
		while (true) {
```

```
                String command = scanner.nextLine();
                String[] commands = command.split(" ");
                String commandType = commands[0];


                switch (commandType) {
                    case "SHOW":
                        if (commands.length == 1) {
                            expenseManager.showBalances();
                        } else {
                            expenseManager.showBalance(commands[1]);
                        }
                        break;
                    case "EXPENSE":
                        String paidBy = commands[1];
                        Double amount = Double.parseDouble(commands[2]);
                        int noOfUsers = Integer.parseInt(commands[3]);
                        String expenseType = commands[4 + noOfUsers];
                        List<Split> splits = new ArrayList<>();
                        switch (expenseType) {
                            case "EQUAL":
                                for (int i = 0; i < noOfUsers; i++) {
                                    splits.add(new
EqualSplit(expenseManager.userMap.get(commands[4 + i])));
                                }
                                expenseManager.addExpense(ExpenseType.EQUAL, amount,
paidBy, splits, null);
                                break;
                            case "EXACT":
                                for (int i = 0; i < noOfUsers; i++) {
                                    splits.add(new
ExactSplit(expenseManager.userMap.get(commands[4 + i]), Double.parseDouble(commands[5 +
noOfUsers + i])));
                                }
                                expenseManager.addExpense(ExpenseType.EXACT, amount,
paidBy, splits, null);
                                break;
                            case "PERCENT":
```

```
                                    for (int i = 0; i < noOfUsers; i++) {
                                        splits.add(new
PercentSplit(expenseManager.userMap.get(commands[4 + i]), Double.parseDouble(commands[5 +
noOfUsers + i])));
                                    }
                                    expenseManager.addExpense(ExpenseType.PERCENT, amount,
paidBy, splits, null);
                                    break;
                            }
                            break;
                    }
                }
            }
}
```

The current design is extensible to support optional requirements #2 and #3 as well. For optional requirement #4, I could call a method ‘simplify’ at the end of addExpense which would simplify the balances based on the current state.

So, this is how I would have designed Splitwise during an actual machine coding round for the given problem statement. This may not be a perfect solution but I guess it would be good enough to clear the interview given that there are only two hours to solve this.