VISHNU INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Approved by AICTE, Accredited by NBA & NAAC and permanently affiliated to JNTUK)

BHIMAVARAM- 534202

A Project report submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

Submitted by

1. **Golla Asreetha Sreeja, Reg.no:22PA1A4240**
2. **Korukonda Keerthi Sujana, Reg.no:22PA1A4257**
3. **Kadurla Yasodha, Reg.no:22PA1A4246**

TITLE - BILL SPLITTING APPLICATION.

Introduction :

What is Split Billing?

Split billing is a method of dividing expenses among individuals, ensuring everyone pays their fair share based on their contribution or usage. It eliminates the need for one person to cover the entire cost and collect reimbursements later. Instead, it allows for real-time tracking and immediate settlements. This approach ensures transparency, reduces misunderstandings, and simplifies managing shared expenses. Split billing is commonly used in scenarios like group dining, travel, or shared household costs.

Figure-1: An image showing a group of individuals dividing a restaurant bill collaboratively, symbolizing the concept of fair expense sharing.

Problem Statement

Our project aims to develop an easy-to-use system for managing shared expenses within groups. It simplifies tracking of expenses, calculating individual shares, and settling balances. The goal is to ensure fairness and transparency, removing the complexities of managing finances in group settings. The system automatically calculates debt settlements, ensuring accurate and efficient handling of group expenses. This approach promotes a smooth financial experience for all users, eliminating the need for manual calculations. The overall goal is to make group financial management more efficient and user-friendly.

- **Expense Tracking**: Simplify the process of tracking shared expenses among group members.
- **Fair Calculation**: Ensure accurate and fair distribution of individual contributions based on the expense type.

- **Debt Settlement**: Facilitate efficient and transparent debt settlements between group members.

System Architecture :

1. **User Class**: This class holds details for each person using the system, such as their name and the total balance. It tracks how much money a user owes to others, helping the system monitor individual debts and payments.

```java
package Models;

import java.util.HashMap;

public class User {
    private String userName;

    private Double totalbalance;
    private HashMap<String, User> owedUsers;

    /**
     * @param userName
     */
    public User(String userName) {
        this.userName = userName;
        setOwedUsers(new HashMap<String, User>());
        this.totalbalance = 0d;
    }

    /**
     * @return the owedUsers
     */
    public HashMap<String, User> getOwedUsers() {
        return owedUsers;
    }

    /**
     * @param owedUsers the owedUsers to set
     */
    public void setOwedUsers(HashMap<String, User> owedUsers) {
        this.owedUsers = owedUsers;
    }

    /**
     * @return the userName
     */
    public String getUserName() {
        return userName;
    }

    /**
```

```
     * @return the totalbalance
     */
    public Double getTotalbalance() {
        return totalbalance;
    }

    /**
     * @param totalbalance the totalbalance to set
     */
    public void setTotalbalance(Double totalbalance) {
        this.totalbalance = totalbalance;
    }

    /**
     * @param userName the userName to set
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }

}
```

2. **Expense Class**: This class keeps track of each expense. It stores who paid, the total amount, and how it is split between multiple users. It also records who owes money to whom and the exact amounts.

```
package Models;

import java.util.HashMap;

public class Expense {
    private String paidUser;
    private Double amount;
    private int noOfOwedUsers;
    HashMap<String, Double> userOwedAmounts;
    private String ExpenseType;

    public HashMap<String, Double> getuserOwedAmounts() {
        return userOwedAmounts;
    }

    public void getuserOwedAmounts(HashMap<String, Double> userOwedAmounts)
{
        this.userOwedAmounts = userOwedAmounts;
    }

    /**
```

```
     * @return the paidUser
     */
    public String getPaidUser() {
        return paidUser;
    }

    /**
     * @return the expenseType
     */
    public String getExpenseType() {
        return ExpenseType;
    }

    /**
     * @param expenseType the expenseType to set
     */
    public void setExpenseType(String expenseType) {
        this.ExpenseType = expenseType;
    }

    /**
     * @return the noOfOwedUsers
     */
    public int getNoOfOwedUsers() {
        return noOfOwedUsers;
    }

    /**
     * @param noOfOwedUsers the noOfOwedUsers to set
     */
    public void setNoOfOwedUsers(int noOfOwedUsers) {
        this.noOfOwedUsers = noOfOwedUsers;
    }

    /**
     * @return the amount
     */
    public Double getAmount() {
        return amount;
    }

    /**
     * @param amount the amount to set
     */
    public void setAmount(Double amount) {
        this.amount = amount;
    }
```

```
    /**
     * @param paidUser the paidUser to set
     */
    public void setPaidUser(String paidUser) {
        this.paidUser = paidUser;
    }

}
```

3. **SplitWiseService Class**: This class is the main part of the application. It manages creating users, adding expenses, and dividing expenses among users. It ensures the amount each person owes or is owed is updated correctly.

```
package Models;

import java.util.HashMap;

public class SplitWise {
    private HashMap<String, User> splitWiseUsers;

    /**
     * @return the splitWiseUsers
     */
    public HashMap<String, User> getSplitWiseUsers() {
        return splitWiseUsers;
    }

    /**
     * @param splitWiseUsers the splitWiseUsers to set
     */
    public void setSplitWiseUsers(HashMap<String, User> splitWiseUsers) {
        this.splitWiseUsers = splitWiseUsers;
    }

}
```

4. **Helper Class**: This class offers useful functions, such as generating test data or initializing users. It simplifies tasks like setting up users or creating random data for testing the system.

5. **Balance Calculation & Display**: This step calculates the balance for each user—whether they owe money or are owed. It checks the debts and payments and then shows each user's balance or the group's overall balance.

```
import Services.SplitWiseService;
import Utitlity.Round;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Scanner;
```

```
public class DriverCode {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);

        int inpsSize = 0;
        SplitWiseService splitWiseService = new SplitWiseService();
        while (inpsSize <= 10) {
            String type = sc.next();
            // System.out.println(owerdUsersList);
            if (type.equals("EXPENSE")) {
                String paidUser = sc.next();
                Double amount = sc.nextDouble();
                int noOfOwedUsers = sc.nextInt();
                HashMap<String, Double> owedUsers = new HashMap<>();
                List<String> owerdUsersList = new ArrayList<>();
                int i = 0;
                while (i < noOfOwedUsers) {
                    owerdUsersList.add(sc.next());
                    i++;
                }
                String ExpenseType = sc.next();
                // System.out.println(owerdUsersList);
                i = 0;
                if (ExpenseType.equals("EQUAL")) {
                    Double equalAmount = Round.round(amount /
owerdUsersList.size());
                    owedUsers.put(owerdUsersList.get(0), equalAmount +
(amount - equalAmount * owerdUsersList.size()));
                    i = 1;
                    while (i < noOfOwedUsers) {
                        owedUsers.put(owerdUsersList.get(i), equalAmount);
                        i++;
                    }
                } else if (ExpenseType.equals("PERCENT")) {
                    Double totalPercentage = 100d;
                    while (i < noOfOwedUsers) {
                        Double currPercentage = sc.nextDouble();
                        totalPercentage -= currPercentage;
                        owedUsers.put(owerdUsersList.get(i), amount *
(currPercentage / 100));
                        i++;
                    }
                    if (totalPercentage != 0)
                        owedUsers = new HashMap<>();
                } else {
                    while (i < noOfOwedUsers) {
```

```
                        owedUsers.put(owerdUsersList.get(i),
sc.nextDouble());
                        i++;
                    }
                }
                splitWiseService.expense(paidUser, owedUsers);
            } else if (type.equals("SHOW")) {
                String user = sc.next();
                splitWiseService.show(user);
            } else {
                splitWiseService.show(null);
            }
            inpsSize++;
        }
        sc.close();
    }
}
```

Key Features :

- **User-Centric**: Designed to track and manage individual expenses, allowing easy tracking of debts and payments.
- **Expense Management**: Records and divides expenses among users, ensuring accurate distribution.
- **Interactive**: Users can engage with the system by adding, updating, and viewing their expenses.
- **Data Integrity**: Maintains precise calculations for balances and ensures accuracy in debt distribution.
- **Scalable**: Supports multiple users and expenses, easily adapting to larger groups or more complex use cases.

Implementation Steps :

1. **Define User Class**:
    - Create a **User** class with attributes:

      userId, **name**, **email**, **mobile**, and **balances**.
    - Include methods to add balances and get balances.
2. **Define Expense Class**:
    - Create an **Expense** class with attributes:

      payer, **amount**, **participants**, **expenseType**, and **shares**.

3. **Initialize Application**:
    - Create a list to store users.
    - Create a dictionary to track balances between users.
4. **Add User**:
    - Input user details (**userId**, **name**, **email**, **mobile**).
    - Create a new **User** object and add it to the user list.

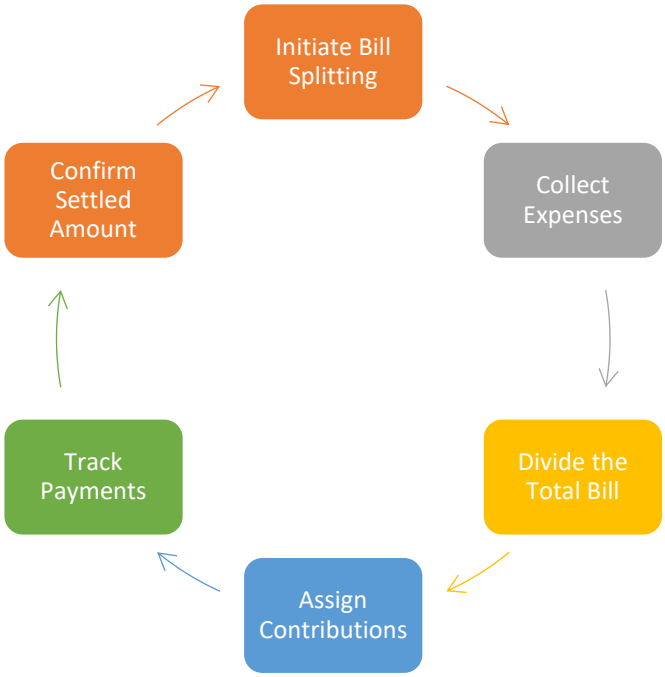

Figure-2: Sequential flow of the bill-splitting process

5. **Record Expense**:
    - Input details for the expense (**payerId**, **amount**, **participantsList**, **expenseType**, and optional **shares**).
    - Validate the expense type:
        - **EQUAL**: Split the amount equally among participants.
        - **EXACT**: Ensure the total of shares matches the amount.
        - **PERCENT**: Ensure the total percentage equals 100.
    - Update balances for each participant based on their share.
6. **Show Balances**:
    - Input an optional **userId** to show balances for a specific user.

- If **userId** is provided, display how much each participant owes that user.
- If no **userId** is provided, display balances for all users.

7. **Main Loop**:
    - Continuously accept user commands until an exit command is received.
    - Parse commands to add users, record expenses, or show balances.

Example of Commands

- Add User: ADD_USER u1 "John Doe" "john@example.com" "1234567890"
- Record Expense: EXPENSE u1 1000 3 u2 u3 u4 EQUAL
- Show Balances: SHOW u1 or SHOW

Applications :

1. **Group Travel:** Split accommodation, transportation, and meal costs fairly among friends or family.
2. **Roommates:** Track and manage shared expenses for utilities, groceries, and rent.
3. **Dining Out:** Quickly calculate each person's share when dining with others.
4. **Event Planning:** Share costs of organizing events like parties, picnics, or gatherings.
5. **Small Businesses:** For team outings, managing office expenses, or project-based costs.

Figure 3: Essential Features of Bill-Splitting Apps
Source: TopDevelopers.co

Conclusion:

This project successfully demonstrates an efficient system for managing shared expenses among users. It ensures accurate tracking of balances, allowing users to see detailed records of debts and payments. By utilizing clear input commands and producing comprehensive output, the system simplifies expense management in a group setting. The implementation highlights fairness, accuracy, and flexibility in handling different splitting methods, making it a robust solution for shared financial scenarios.