SplitWise

Course Title

SWE 4302

PRESENTED BY

Adid-Al-Mahamud Shazid
210042172

PRESENTED TO

Jubair Ibne Mostafa
Assistant Professor
Department of CSE
Islamic University of Technology

Project Overview

SplitWise is a comprehensive expense management platform designed to simplify the way users handle their personal and group finances. With a user-friendly interface, SplitWise enables users to sign up, add friends, and effortlessly log expenses, whether shared among a group or involving individual friends. The system provides a transparent overview of balances, facilitating easy debt settlement and accurate tracking of financial interactions.

Features

User Management

- Sign Up
- Login
- List of Users

Expense Tracking

- Add expenses
- Multiple user shared Expense
- One user and one friend shared expense

Friend Management

- Add Friend
- Remove Friend
- List of Friends

Balance Tracking

- Calculate and display balances
- Shows who paid how much
- Shows how much each person owes

Code and Design Smells

1	Inconsistent Naming
2	God Class
3	Duplicate Code
4	Data Clumps
5	Black Sheep
6	Shotgun Surgery

1	Rigidity
2	Fragility
3	Viscosity
4	Immobility
5	Needless Complexity
6	Opacity

Application of SOLID

1. Single Responsibility Principle (SRP):

The Main class of un-refactored code had multiple responsibilities such as managing the user interface, handling user input, and interacting with user-related functionalities like login, sign-up, etc. So, I Separated these responsibilities into distinct classes to ensure each class has a single responsibility. In my refactored code:

- **User** class
- **Expenses** class
- Operation classes (**GroupExpenses, NonGroupExpenses, GroupOperations**, **NonGroupOperations**, **ManageFriends**, **UserMenu**, **SignUp**, **Login**, **HomeMenu**)

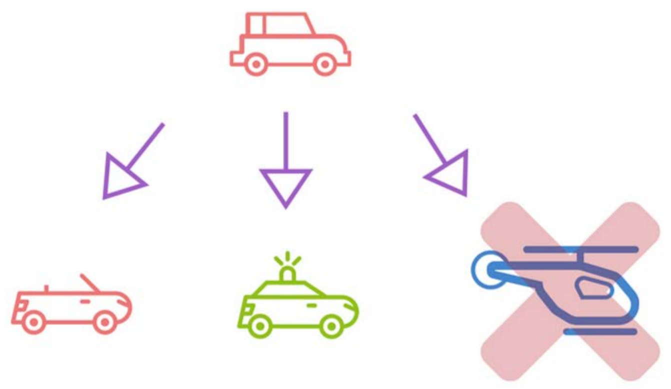
2. Open/Closed Principle (OCP):

In the Main class of un-refactored code, the menu options and related logic are hardcoded within the methods. Adding a new menu option or modifying existing behavior would require changing the existing code. After Refactoring, I can easily introduce new methods in my code.

- **IExpenses** interface
- Operational Classes

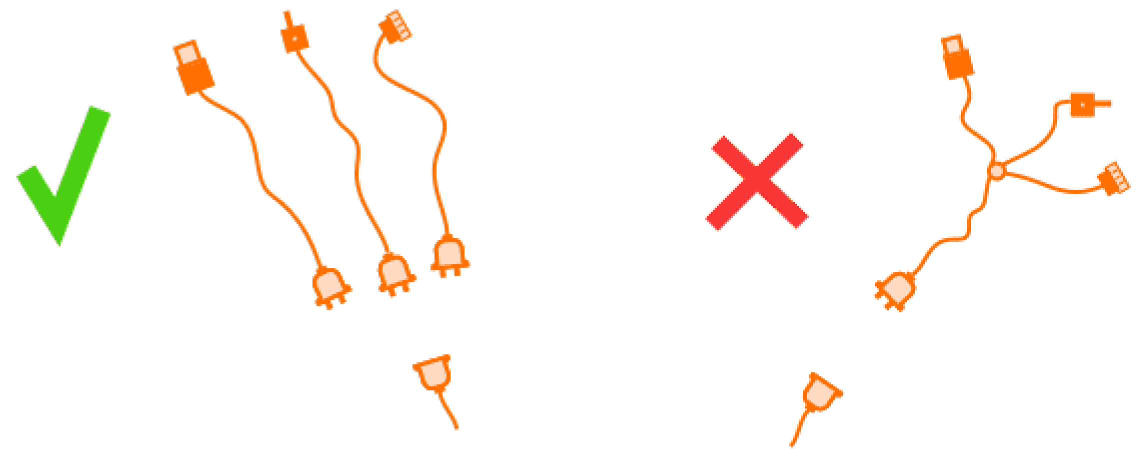
3. Liskov Substitution Principle (LSP):

I didn’t make any use of LSP in my code.



4. Interface Segregation Principle (ISP):

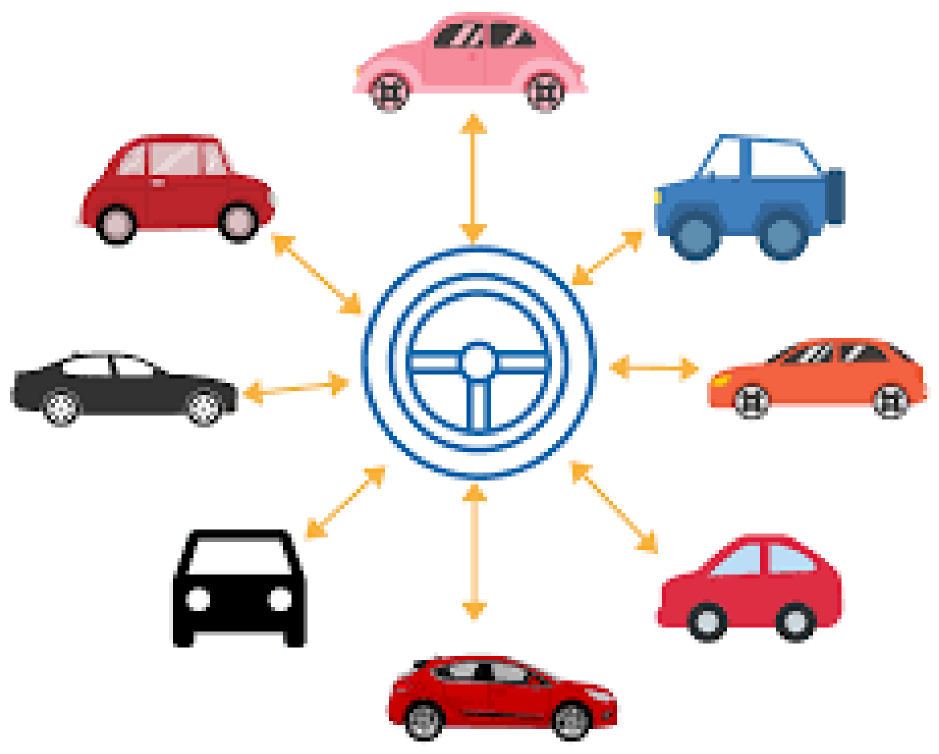
The **IExpenses** interface is used to segregate methods related to expenses. Classes that need to handle expenses implement this interface, and they only need to provide implementations for the methods relevant to their responsibilities (**addExpenses()** and **balances()**).



5. Dependency Inversion Principle (DIP):

The Main class of un-refactored code is tightly coupled, where direct instantiation of concrete classes like Login, SignUp, UserMenu, etc., is happening. So, I consider using dependency injection to decouple classes and promote better testability and flexibility.

- High level Modules are depending on **User** class which is abstraction instead of concrete classes.
- High level modules (**GroupOperations, NonGroupOperations**) are depending on **IExpenses** interface which is also an abstraction.



Thank You