# Object Oriented Programming Project Assignment

# CS F213

**Project No. – 17**

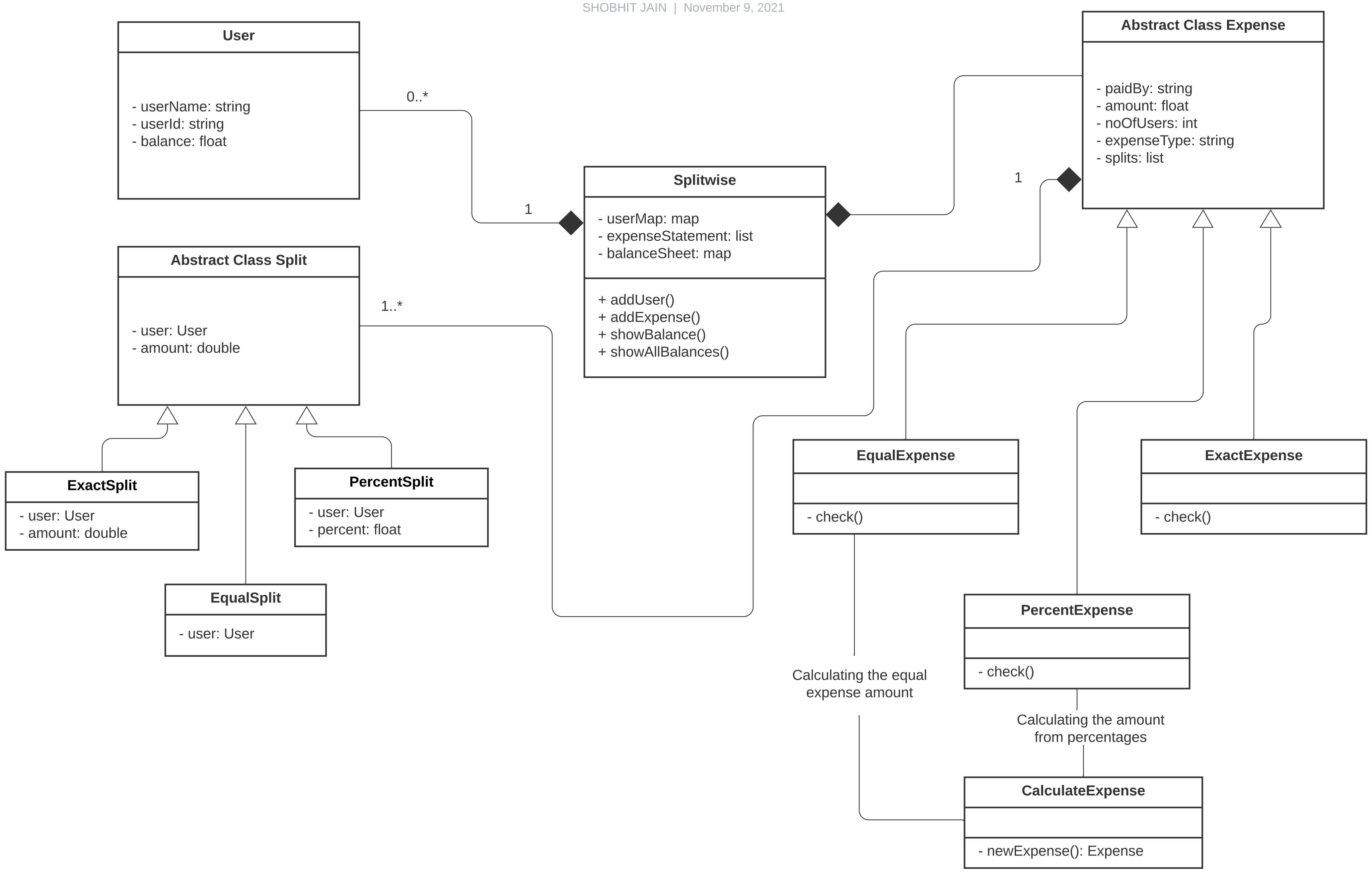
**Project Topic – Splitwise**

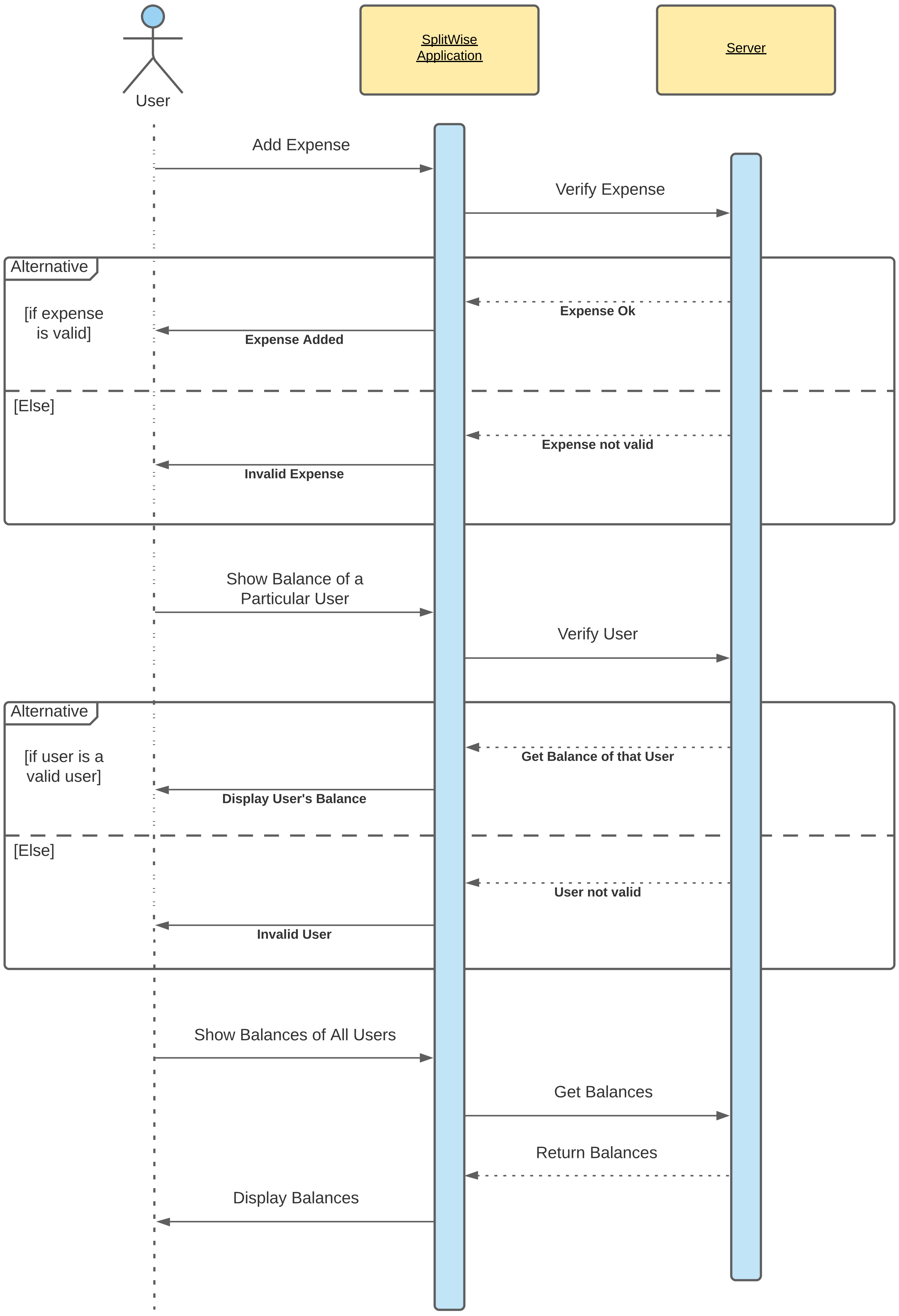
**Submitted By -**

**Shobhit Jain (2019B3A70385P)**

# OOP UML Class

SHOBHIT JAIN | November 9, 2021




User
SplitWise Application
Server
Add Expense
Verify Expense
Alternative
[if expense is valid]
Expense Ok
Expense Added
[Else]
Expense not valid
Invalid Expense
Show Balance of a Particular User
Verify User
Alternative
[if user is a valid user]
Get Balance of that User
Display User's Balance
[Else]
User not valid
Invalid User
Show Balances of All Users
Get Balances
Return Balances
Display Balances

**1. Factory Route:**

It is our plan within the split wise class to separate costs between users we need

create different types of different classes like EQUAL, EXACT, PERCENT and use

same we used different loops if otherwise something within our main dividing class. To avoid this

We could use a factory design pattern and we should have created one creator phase

(class “splitcreate”) and one product class division () which returns a user-based classification category input, i.e. if users want to divide the cost equally an equally divided item should be returned again if the user wants to split the cost A clever percentage rather than a percentage difference should be divided. But now something new implementation will lead to the violation of one of the principals '' Open extension is closed to be repaired ’.

So, we can use the factory method in our system to create the conditions for different classes

to make our code work better.

**2. Encapsulation:**

The whole idea behind encapsulation is to hide usage details for users. If the data member is private it means it can only be accessed in the same category. No external class can access the private (flexible) data member of another class.

However, if we set the public getter and setter methods to update (for example void setSSN (int ssn)) and read (for example int getSSN ()) the private data field the external category can access those private data sources through public channels.

In this way the data can only be accessed through public channels thus making private fields and their operations hidden in external classes. That is why encapsulation is known as data encryption. Let us look at an example to better understand this concept.

**3. Singleton Pattern:**

This pattern can be used for the implementation of our program. We can use this pattern to

use the cost manager section as we require one example of cost manager once

a global access point and we do not want users to build multiple class objects / models

"Cost manager" because if a lot of cost management equipment is built that can lead to

creating too many balance sheets, which will create confusion and can lead to the addition of

incorrect records in our system. Applying this pattern can be very helpful in using

GUI as it is very easy to keep all records if there is only one balance (expense manager

object) is created.

**4. Strategy pattern:**

We did not apply this pattern directly but used the asset to use the separator

behaviour, we created class divisions and extended class 3 from class division.

If we had read this pattern before we would have used our code in this way

create a cost class with data members such as who paid, user number, user list

the amount due, and the type of expense (in the present case equal, direct, percentage) etc.

. then we would have created classes for inherited costs, and we would have been able to

separate the same things from the different ones as different behaviours from all

The cost class should have been used separately. so that we can use a

interface split behaviour. Three practical strategies for differentiated behaviour 1. Equal segregation 2. Absolutely split by 3 percent

This way we could apply our code using the strategy pattern and follow some of them

the head of the oops as the visible disconnection, the behaviour of the integrated isolation.

**5. Decorative pattern:**

This pattern is not good for this project.