For input, print all the options along with numbers and take numeric inputs (if possible).

Observer Pattern

ABC Company provides server as a service to its clients. But as the company is very new and small, currently they have only one server. So, when maintenance or upgrading work is carried out, the server maybe partially or fully down. The company has two types of clients: **premium user** and **regular user**. Premium users have to pay more than the regular users and in return, they receive some benefits. As a part of an agreement, all the data of the premium users are backed up in a server of DEF Company. When the server of ABC Company is partially down (some functionalities are not available) or fully down, DEF Company helps to provide an uninterrupted service.

Previous State of server	Current State of server	For Premium users	For Regular users
Operational	Partially down	ABC company asks the user whether he/she wants to use service from two servers (partially from the server of ABC and partially from the server of DEF) or from one server (DEF)	ABC company asks the user whether he/she wants to continue using the limited functionality or pay $20 per hour to enjoy the full functionality taking service from server of DEF (this will copy all his/her data to the server of DEF)
Operational	Fully down	ABC company notifies the user about the fact that the service is now provided by their partner DEF company	ABC company asks the user whether he/she wants to pay $20 per hour to take service from DEF company (this will copy all his/her data to the server of DEF)
Partially down	Operational		If the user was paying, send him/her the total bill since the last state change. *
Partially down	Fully down	If the user was taking service from two servers, ABC company shifts all the services to the server of DEF and notifies user about it.	
Fully down	Operational		If the user was paying, send him/her the total bill since the last state change. *
Fully down	Partially down		

In addition to the notifications stated above, ABC company informs all the users of any state change.
You may need to assign state to the users as well.
***You don't need to calculate the total bill. Just use some arbitrary placeholder variable like x.**
For simplicity, create one instance of premium user and one instance of regular user. Sending notifications from the ABC company to these two users will suffice.
Code in Java using Observer Pattern to create the above scenario.

Input-Output:
Start with Operational state of the server. Take input to change the state of the server (for example, input "1" to change the state to Partially Down or "2" to change the state to Fully Down). You should be able to change the state of the server from every state to every other state from input. Notifications will be printed from the side of users (premium or regular) as prompts, i.e. "Do you want to use service from two servers?" User will select an option and the service will continue accordingly. Notification of data copying should also be printed from the user side. Communication from the users to ABC company need not follow any specific pattern, but the reverse communication should follow the observer pattern.

Mediator Pattern

Exam controller office plays an important role in arranging exams. In a university, the examiners send all the exam scripts and corresponding marksheets to the exam controller office after checking the exam scripts. The exam controller office scrutinizes the marks and correct mistakes (if any). Then it publishes the results to the students. If a student thinks that he/she should have got more marks in an exam than he/she actually got, he/she can apply for re-examine. The exam controller office then sends the corresponding exam script to the corresponding teacher. The teacher re-examines the script and let the exam controller office know if there is any change (increase or decrease) in marks. The exam controller office then lets the student know the results of re-examine and update the mark if there is any change.
Code in Java using Mediator Pattern to create the above scenario. Consider exam controller office as the mediator.

Input-Output:
For simplicity, consider 5 students and an examiner. Assume that the examiner has already checked the exam scripts of the 5 students. The examiner sends the exam scripts of them and corresponding marks (as a list) to the exam controller office (i.e. print "scripts and marks of student id 1,2,3,4,5 sent to exam controller office"). Print all the student IDs and corresponding marks from the exam controller side after exam controller office gets the documents. Set the probability of mistake in marking to 0.5 or 0.6 and randomly generate cases of mistakes. Make

sure that at least one case of mistake is generated in a total of 5 cases. Print the case of mistake mentioning student ID, previous mark and corrected mark. Now publish the results to the 5 students. Instances of "Student" class will print their individual mark now.
In case of re-examine, take input to generate a re-examine request. The input will decide which student will apply for re-examine. The request should go through the student side (i.e. print "re-examine request sent from student id 1") and should be printed from the exam controller side (i.e. print "re-examine request got from student id 1"). Same type of communication should be followed elsewhere.

Submission instructions:

- Enclose all the files inside a single folder named after your 7 digit student id. For example, 1605001 will enclose all the files in a folder named 1605001. Then, zip the folder (the name of the zipped file will also be the same as the folder, i.e., 1605001.zip).
- Submit the zipped file in moodle by 23 August 2019 (**10 PM**).
- A submission at the sessional class won't be accepted if there is no submission in moodle.