Design Patterns

CS 121, second year students

Assignment 1

September 04. Due to September 18.

Overview

In this assignment you have to implement the associations and aggregations between given classes. Basic knowledge in object oriented design is required. You may use arbitrary programming languages which support OOP. Do not forget to properly comment your code.

Description

We are all living in a world where technological revolutions occur almost every day. One of the most important revolutions was, of course, the Communication Systems. Today, you're going to implement a simple version of a communication system. There will be Customers, Operators, and Bills that belong to the Customers. Customers can talk to each other, message each other, or they can connect to the internet via their mobile phones. Operators have their unique costs for the corresponding actions that will be given to you in the input. Every customer will have their unique Bill that stores and does the necessary actions about their spending.

Note that there will be multiple classes. Therefore, think a little bit about the structure before starting to implement the program. Field and Method names that are given to you in this Document must be identical in terms of names and structures in your Project. That doesn't mean that you can not add extra Methods or Fields.

Classes and Implementation Details:

There will be 4 classes interacting with each other in this Project: **Main**, **Customer**, **Operator** **Bill**. Note that, it will be better to do the necessary calculations via using corresponding methods in the Customer, Operator or Bill class, not in the main Class (which will run everything). Some parts of the main class are given to you (input – output operations) but, other 3 classes are given completely empty. You're expected to fill these classes with the given directions.

Main Class

Is used for the general input - output operations. You are going to read an input file (JSON or hard code everything directly) that contains directions about the simulation and do the several actions, then you will print out the desired results. These directions will be explained in a detailed way, later on this document. Name of the input file will be given as arguments of the program. For the input-output test cases, we are going to use your main class. There will be 3 arrays in the main method as follows:

Customer[] customers

Operator[] operators

Bill [] bills

You are going to store the corresponding data in these arrays. All of these arrays must be initialized and all of the elements in these arrays must be set to "null" as default. You're going to edit these arrays whenever a new object is created.

Class Customer

Customer Class must have the following fields, exactly as named below:

- **int ID**
- **String name**
- **int age**

- **Operator [] operators**
- **Bill [] bills**

Customer class must have the following methods with the exact parameters:

- A constructor with five parameters: int ID, String name, int age, Operator[] operators, Bill [] bills, double limitingAmount.
- ***void talk(int minute, Customer other)*** for customers to talk via the operator. Other is another customer, mainly the second customer.
- ***void message(int quantity, Customer other)*** for customers to send message, amount is the number of messages to be sent. Other is another customer, mainly the second customer.
- ***void connection(double amount)*** for customers to connect the internet. Amount is the number of data as MB.
- Getter and setter methods for age,operator,and bill. Ex: getAge (), setAge(int age)

Class Operator

Operator class should have these fields with the exact names:

- **int ID**
- **double talkingCharge**
- **double messageCost**
- **double networkCharge**
- **int discountRate**

Operator class should have the following methods with the given parameters:

- A constructor with 5 parameters: ID, talkingCharge, messageCost, networkCharge, discountRate
- ***double calculateTalkingCost(int minute, Customer customer***) for calculating the total amount to pay for talking.

- ***double calculateMessageCost(int quantity, Customer customer, Customer other)*** for calculating the total amount to pay for talking.
- ***double calculateNetworkCost(double amount)*** for calculating the total amount to pay for talking
- Getter and setter methods for talkingCharge, messageCost, networkCharge, discountRate.

Class Bill

Bill class must have the following fields:

- **double limitingAmount**
- **double currentDebt**

Bill Class must have the following methods with the exact parameters:

- Constructor with 1 parameter: limitingAmount. Note that you should initialize the currentDebt as zero.
- ***boolean check(double amount)*** is for checking whether the limitingAmount is exceeded or not.
- ***void add(double amount)*** is for adding debts to the bill.
- ***void pay(double amount)*** is for paying the bills with the given amount.
- ***void changeTheLimit(double amount)*** this method is for changing the limiting Amount.
- Getter methods for limitingAmount and currentDebt.

Input format

You must set the number of customers N and the number of operators M (M!=N). Then the next operations must be implemented and tested:

1. Creating a new Customer;
2. Creating a new Operator;
3. A customer can talk to another customer;
4. A customer can send a message to another customer;
5. A customer can connect to the internet;
6. A customer can pay his/her bills;
7. A customer can change his/her operator;
8. A customer can change his/her Bill limit.

Instructions:

Most of the calculations must be in the **Customer, Operator** and **Bill** classes and in their methods. For the talking charge, charge must be the amount of minutes times the operator's talking charge per minute. If the Customer's age is below age 18 (18 is excluded) or higher than 65 (65 is excluded), then the operator applies a discount with the corresponding discount rate that is given.

For messaging, cost must be the quantity of the messages times the operator's message cost per message. If the 2 customers are using the same operator, then the operator applies a discount with the corresponding discount rate that operator has.

For internet usage, calculations are straight forward, amount of MBs * the network cost per MB. Before the actions, you should check the limit of the bill, if someone would exceed the limit after the actions, then no actions must occur.

We assume that every customer has enough money to pay their bill for the desired amount. ID of the Customers, Operators, and Bills should start from 0.

You can declare extra fields or extra methods. But the given ones in this document must exist in your program and their parameters and names must be the same with the ones that are given.

References:

***Scott W. Ambler*. The Object Primer 3rd Ed: Agile Model Driven Development (AMDD) with UML 2.**

***Robert Martin*. Clean Code: A Handbook of Agile Software Craftsmanship.**

***Robert Martin*. Agile Software Development, Principles, Patterns, and Practices.**

https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

https://www.learncpp.com/cpp-tutorial/welcome-to-object-oriented-programming/

https://realpython.com/python-property/

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

https://www.w3schools.com/python/python_dictionaries.asp

https://docs.python.org/3/tutorial/datastructures.html#