CTS 02.04.2024

Number 1

Name:
Group Number:
Grade: ☐ /10p

Developing a software application for managing payments of a private hospital.

4 points: A private hospital wants to implement an application to keep track of admissions, considering the following cases for each patient: whether all costs will be reimbursed by the National Health Insurance House or by Private Insurers or if they will be paid by the patient from their own funds. Since the waiting time for the Private Hospital to receive money from Private Insurers is longer by 1 month than if the patient were to pay from their own funds, and receiving money from the National Health Insurance House takes over 3 months, the hospital management wants to offer discounts on all procedures depending on the type of payment selected by the patient (they can only select one option). The selected payment type will be received as an input parameter of type enum.

The discounts are the following:

- Patient Funds: 20%
- Private Insurers: 10%
- National Health Insurance House: 0%

The implementation will be carried out starting from the IPayment interface attached to this statement.

1 point: In main(), a relevant code sequence must be implemented to highlight the design pattern implemented. After selecting the payment method, the hospital system must connect to a payment service to initiate the transaction to the National Health Insurance House/Private Insurer/Client Bank. The hospital enters into a contract with the EuPlatesc.ro service for payment.

4 points: Within the implemented application, a payment initiation module will be created. Due to licensing reasons, once a payment component is created, it will be used for the transactions of all patients. Another payment entity cannot be created. The implementation will be carried out through the implementation of the IPaymentProvider interface.

1 point: In main(), a relevant code sequence must be implemented to highlight the design pattern implemented.

```java
public interface IPayment {
   public double getDiscount();
}
```

```java
public interface IPaymentProvider{
   public void initiateTranzaction();
}
```

OBS:

1. The implemented patterns must respect the definition and structure discussed in the courses and laboratories. Variations or incomplete implementations are not accepted.
2. The pattern must be implemented correctly and completely (in its entirety) to be considered.
3. The solution does not contain compilation errors.
4. Patterns can be treated separately or can be implemented on the same set of classes.
5. Implementations that are not functionally related to the requirements in the subject will not be considered (copying an example from other sources will not be scored).
6. Modifying the classes received is *not* allowed.
7. Solutions will be cross-checked using MOSS. Sharing code between students is not allowed. Solutions with a similarity degree higher than 30% will be canceled.

Mandatory Clean Code Requirements (the solution is deducted by 2 points for each requirement not respected) - up to 8 points may be lost.

1. For naming classes, functions, attributes, and variables, the Java Mix CamelCase naming convention is followed as discussed.
2. Patterns and the class containing the main() method are defined in distinct packages with the form
    a. cts.firstname.lastname.groupNumber.pattern_name,
    b. cts.surname.name.groupNumber.main (students in the additional year replace groupNumber with "as").
3. Classes and methods are implemented respecting the principles of KISS, DRY, and SOLID (pay attention to DIP).
4. Class names, method names, variables, as well as messages displayed in the console, must be related to the subject received (generic names are not accepted). Functionally, methods will display messages in the console to simulate the required action or will implement simple processing.