HMS with billing component

Putting Polymorphism into practice!

Let’s now extend the HMS system with Billing Component, which would be responsible for billing a patient. Patient & User classes are needed for this exercise. In this exercise, we will put **polymorphism** concept & **instanceof** **operator** into practice. This exercise should give you a better feel on how nicely object-oriented programming models real-world scenarios.

When a patient is being billed after treatment, hospitals would apply any *insurance* that the patient may be having. That way the insurance company would pay a part of the medical bill and the remaining will be paid by patient.

Let’s assume we have the following four classes that represent four insurance plans and a patient can buy one of them:

**PlatinumPlan** - the highest coverage of 0.9, i.e., it covers 90% of the total medical expenses and the patient would pay the remaining 10%.

**GoldPlan** - the coverage of 0.8, i.e., it covers 80% of the total medical expenses and the patient would pay the remaining 20%.

**SilverPlan** - the coverage of 0.7, i.e., it covers 70% of the total medical expenses and the patient would pay the remaining 30%.

**BronzePlan** - the coverage of 0.6, i.e., it covers 60% of the total medical expenses and the patient would pay the remaining 40%.

These four classes would extend from another class called **HealthInsurancePlan**.

Coverage is indicated by a double field named ‘coverage’ and would be part of HealthInsurancePlan class. Corresponding getter (getCoverage) setter (setCoverage) should also be provided for 'coverage'. Making 'coverage' and its getter setter part of HealthInsurancePlan and not part of its sub-classes would help avoid code duplication (a benefit of inheritance). However, specific sub-classes (e.g. PlatinumPlan) would set the appropriate value for the coverage field.

Next, you need to add a new field called ‘insurancePlan’, which is of type HealthInsurancePlan to the Patient class (implemented in previous exercise) and leave this field uninitialised, i.e., gets a default of null. This field would indicate the insurance plan that a patient has. So, it indicates a HAS-A relationship, which was mentioned in the lecture that discussed IS-A Test. This field 'insurancePlan' would be set by setter (setInsurancePlan) and has corresponding getter (getInsurancePlan). Getter would have a return type of HealthInsurancePlan while setter would have a parameter whose type is HealthInsurancePlan. As you can see, setInsurancePlan(HealthInsurancePlan) would give us the polymorphism benefit as we can pass an instance of any of the above 4 classes.

Next, you would implement the billing logic, which goes into a new class called **Billing.java**, which would have a single static method called **computePaymentAmount**. Template code is provided for this class. Input to this method is a Patient object & 'amount', which is a double value indicating the amount the patient is billed before applying insurance. The method returns a double[] and its first element would have the amount that the insurance company would pay while second element would have the amount that patient has to pay. As mentioned above, you need to make use of 'coverage' of the insurance plan. Note that some patient's may not have any insurance plan in which case 'patient.getInsurancePlan()' in the below code would return a null. Once the patient's part is computed, additional discount should be applied on the patient's part depending on their insurance plan, which is as follows:

PlatinumPlan: $50 discount

GoldPlan: $40 discount

SilverPlan: $30 discount

BronzePlan: $25 discount

If the patient does not have any insurance plan, then a discount of $20 is applied.

As an example, if the patient uses Platinum Plan and if the total amount billed is $1000, then after applying the coverage of 0.9, patient's part would be $100. Up on applying the discount of $50 for Platinum, the final amount that patient must pay would be only $50.

Thanks & Happy Coding!!