AIN3005 - Homework 1 Documentation

- Course Name: Advanced Python Programming
- Course Code: AIN3005
- Author: Atena Jafari Parsa

Introduction

This homework assignment aims to create a simple system for an insurance company that has 3 types of insurance policies; health policy, auto policy, and life policy. The information of the customers are stored privately, and each policy is calculated according to different criteria of the customers' information.
The users (whose commands will be stated in the main function), will only be able to access the existing premiums calculated (accessed by quote generation). The list of the past policies of each customer will also be stored in a list and will be accessible to the users.
In order to show how the designed solution generally works, the UML diagram is shown on the upcoming page to clarify the usage of classes and their attributes and the methods, and the relationships between the classes. Additionally, a general walkthrough of the system is written below.

General Walkthrough of the Designed System

The insurance premium calculator consists of 5 classes in total. The Customer class includes customer details and a list of past policies (past_policies). The Customer class has a composition relationship with the Policy class which is the parent of 3 child classes; HealthPolicy, AutoPolicy, and LifePolicy.

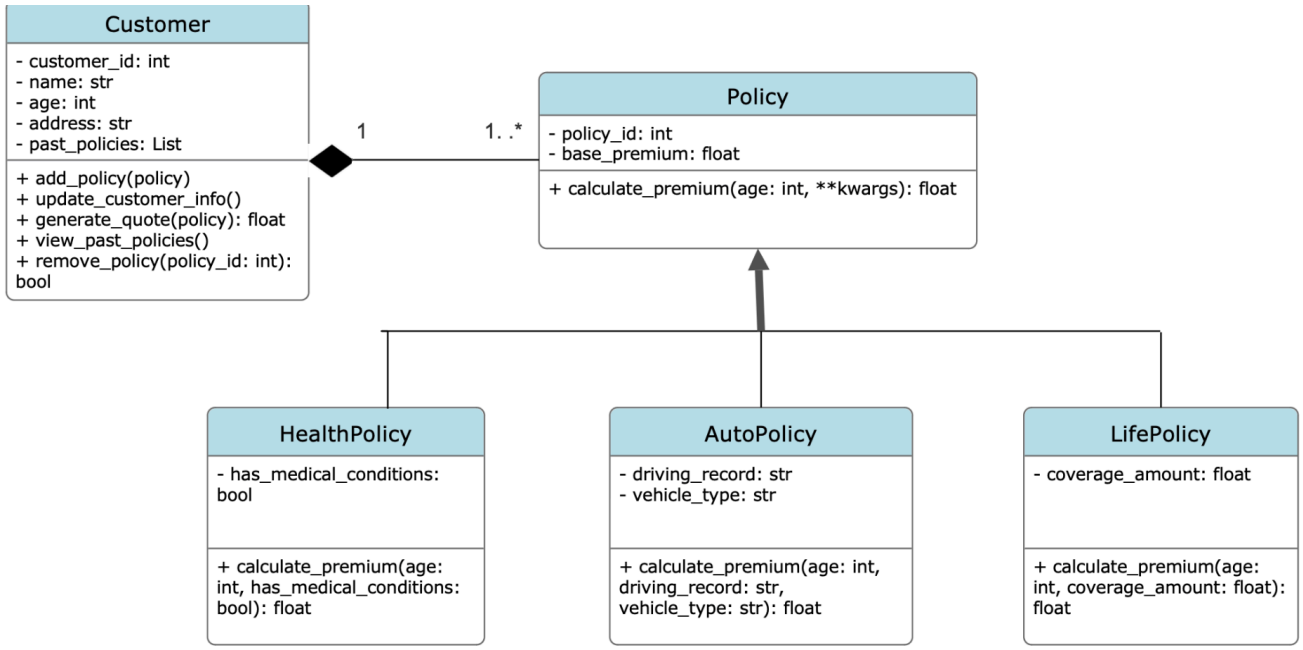
The Customer class provides an add_policy method, allowing customers to select and add different policies to their history, while view_past_policies displays all previous policies. New customer information can be added (add_customer_info())

And the existing information can be updated (update_customer_info()).

1 or more policies can be associated with a customer. Composition between Customer and Policy classes means a policy instance cannot exist without a customer as each policy is created for a specific customer and reflects their insurance details.

The subclasses of Policy inherit the same attributes (policy_id and base_premium) and method as the superclass ,however, the method calculate_premium is overwritten in each subclass. That is why the parameters of the method is written as “age: int, **kwarsgs”. Age will be used in every definition of calculate_premium but the other parameters and the definition of the method will change in the subclasses. More details such as how the premium is calculated in each subclass will be provided later in the documentation.

- The UML Diagram



- Customer Class

The Customer class has 4 attributes. All of them are private Customer_id (integer) is automatically incremented every time a new Customer instance is created to ensure uniqueness of the id. The name has a data type of string, like address. Age is an integer. A list of past policies is also stored and every time the

add_policy method is called, the new policy will be appended to the past_policies list. As shown in the UML diagram, customer information (name, age, address) can be updated by the update_customer_info method. The generate_quote method, prints the name of the customer, the policy type and returns the calculated premium. The view_past_policies method uses a for loop to print all policies with the auto-incremented policy_id and the policy type as well as the calculated premium of the respective Customer instance. A policy can be removed by its id as an argument by using the remove_policy method. If the policy removal is successful, it returns True and prints that the selected policy has been removed, else, it prints the policy id was not found and returns False. As shown in the diagram, all methods of the Customer class are public as they are all meant to assist user accessibility.

- Policy Class

The policy class has an auto-incremented policy_id (to ensure uniqueness) and a base_premium of type float. Both are private. The only method of this class is calculate_premium which has age: int as its argument and **kwargs meaning that the other arguments and the definition of the method will be overridden in the subclasses of Policy. But it would always return the calculated premium as a float. The method is public for user access to premium.

- The Relationship between Customer and Policy Classes

Before exploring the subclasses, let us take a look at the relationship between Customer and the Policy classes. As shown in the UML diagram, They have a composition relationship shown by a filed diamond towards Customer. Because a policy instance is created according to the details and the information of the respective customer instance and the policy cannot exist without a customer. Also, a policy instance can only belong to one customer. However, there may be a customer without a policy or with multiple policies.

- The Subclasses of Policy, HealthPolicy

The HealthPolicy class inherits the attributes (policy_id and base_premium) and the method from its parent, Policy. So by using `super()` before its constructor, we ensure its inheritance. The extra attribute that HealthPolicy takes is a boolean called has_medical_conditions. In order to calculate the health policy premium, the method calculate_premium is defined as such, first, the inherited base premium is assigned as an initial value, then, if the age is more than 60, the

premium will increase by 50% as seniors are usually more likely to need health insurance. If the customer has medical conditions, the premium will increase by 30% as people with medical conditions would naturally require a more intense and expensive health care. After the premium is calculated, the float is returned.

- The Subclasses of Policy, AutoPolicy

AutoPolicy is the other child class of Policy. Just like HealthPolicy, it also inherits the policy_id and the age attributes, and the calculate_premium method. However, it takes new attributes; driving_record and vehicle_type which are both strings. The premium is calculated as such, after applying the base premium, if the age is less than 25, the premium will increase by 50% as young drivers tend to get into more accidents. If the age is more than 60, the premium will increase by 20%. The driving record can be major, minor or none. If it is major, the premium will be multiplied by 2, if it is minor, it will increase by 30%, and if it is none, the premium will not increase in this step. The vehicle type can be luxury, motorcycle, SUV, truck and other. In each case the premium will increase respectively: 20%, 50%, 40%, 60%, 30%. Then, the calculated premium is returned.

- The Subclasses of Policy, LifePolicy

Again, as a child class of Policy, LifePolicy inherits policy_id and base_premium attributes and the calculate_premium method. It takes an extra attribute called coverage_amount: float. It overrides the calculate_premium method like this, After applying the base_premium, it adds it to 1% of the coverage_amount. Then, if the age is more than 65, the premium is added by 30%. Then the premium is returned.
As shown in the UML diagram, all the attributes of the subclasses are private and the calculate_premium method is always public.

- Testing

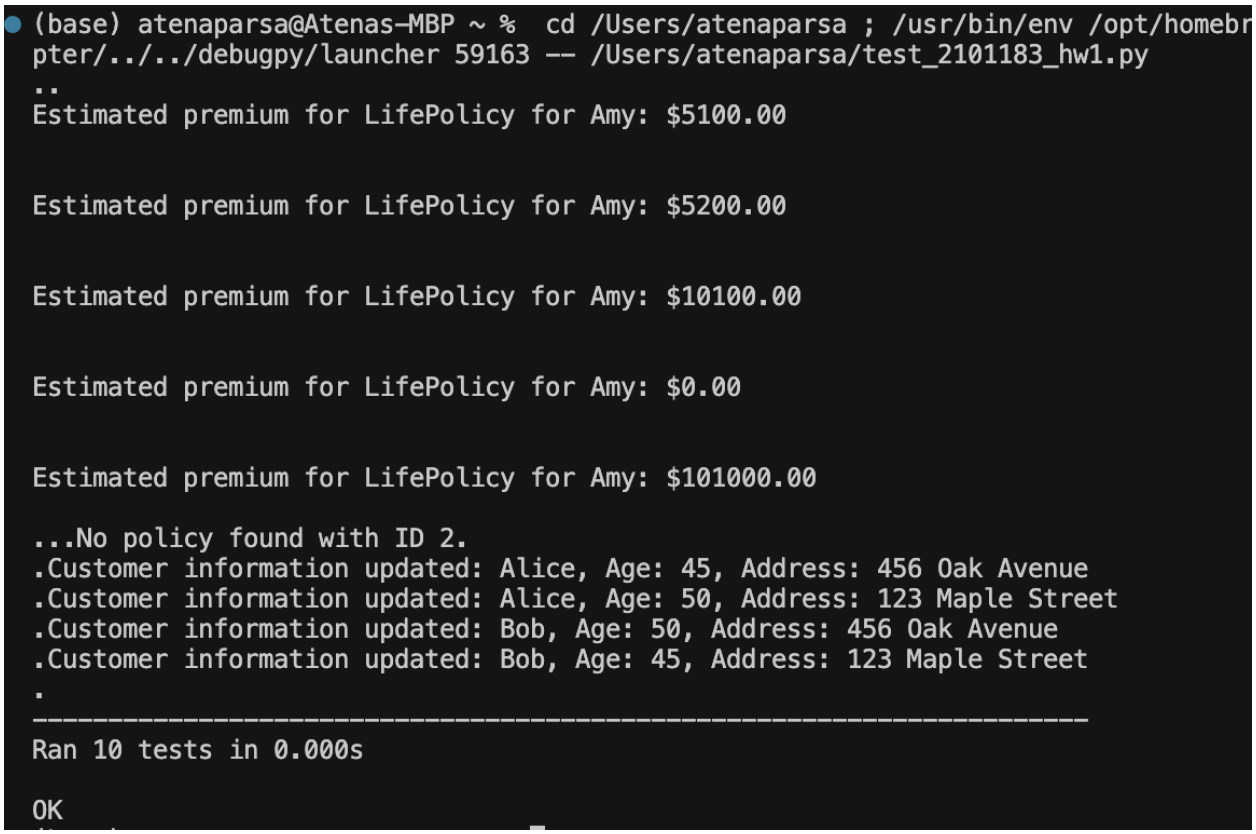
All the unit test cases are performed in test_2101183_hw1.py. A main function has also been written in the main python file (hw1_2101183.py) but it only provides a basic interface for the user through the console so the general structure of the system can be viewed. The main testing as stated above, is done in the unit test cases.

- All 3 policies have been tested by asserting whether the premium calculated by the calculate_premium method equals the actual value (the

actual value was found by the author using mathematical operations and a digital calculator.)

- Adding a policy to a customer instance was tested through test_add_policy_to_customer. Like before, the necessary sample data is provided and then, it is asserted that the policy has been added to past_policies list and the added policy equals the first element of that list.
- Updating all 3 details of the customer instances one by one, and all at once was tested through 4 unit cases (name, age, address, all) by asserting the updated attribute being equal to the new value, not the old one.
- The removal of a policy was tested by asserting that the length of the past policies list has become one less.
- The generation of a quote with an example policy, was tested with minimum (0) and maximum values (reasonably high) for a LifePolicy instance for the base_premium and the coverage_amount.
- Viewing the past policies itself, was also tested by asserting the policies were the same as the ones stored in the policy_list.

Thus, 10 successful tests were run after a few attempts.



One of the challenges I faced was finding a way to write unit test cases for calculating the premiums for each policy. So I calculated the correct values myself and asserted they were equal to what the calculate_premium method calculated.