Marcos Cruz

INSC484

Take Home Project 3 Software Design

11/24/2024

Interface/Main

I’m happy that the main is completely devoid of database calls, all those happen within the Models. The only function calls in the main are all the menu options, some additional input calls to get the required information to make the objects, and the function calls to make and utilize class instances. Adding a client is simple since it just takes basic information that the user types in and then a client method is called to store that information in the database. Same thing with adding CPAs and assistants. The prompt\_client\_materials() function asks for a client ID so that the proper client data can be retrieved and a class instance can be created. There’s just a bunch of if-statements that check whether provided\_materials is True or False, and output a meaningful statement. Prompt\_tax\_return() is similar as it’ll ask for a client’s id so that proper tax return is updated.

Client

The Client class contains the same attributes as the Client table within the database. I overrode the \_\_repr\_\_ function so that during testing I could make sure that the correct values are being assigned to the correct attribute. The save instance method is where Clients get inserted into the database. During this function call, that specific Client’s tax return will also be created. It will mostly contain NULL values until a CPA or Assistant file the tax return. That’s when the timestamp and fileStatus get added/changed. The assign\_cpa instance method gets called right before the save method because all it does is assign a cpa to a client, which is something that is required. The check\_return\_status instance method is the only somewhat complicated function. When it gets called from the main, first it makes sure that the client\_return object isn’t empty, if it is then it’ll just send back an error message. If the object contains data, then it’ll check the fileStatus to see what message it should spit out. If fileStatus is False, it’ll state the Clients name and that they haven’t filed a tax return. If fileStatus is True, then it’ll state which Client turned it in and when. Additionally, it’ll check the CPA\_check attribute, which will determine if the program states that an Assistant filed it and it hasn’t been CPA checked or if it has been filed directly by a CPA. Maybe I could have made these checks more general within the Class, so that the main can handle more custom messages. I made provided\_materials a private attribute because I just wanted to practice with setters. This could easily have been a simple method that just updates the value of provided\_materials and then saves the update, but I wanted to add this extra layer of control within the program. When the main sets provided\_materials to True, it will automatically get a database connection and store this update. There is only one class method because this program only needed to get one client at a time, not multiple. All you need to pass to this class method is a client ID and it’ll receive that client’s information from the database and then create an object for that client. After this getter is used, the submit\_tax\_return() check is called to see whether that client has filed their tax return.

CPA

For this I also overwrote the \_\_repr\_\_ function but not the \_\_str\_\_. It has a very similar save instance method to the Client class as it just inserts that data into the database and assigns the CPA\_id to the instance. There’s a class method here as well that just gets a CPA’s information from the database and returns a CPA object, nothing crazy. All the getters have a try-except block because I was getting AttributeErrors when I tried to create a CPA object with a Nonetype object, this was because I was looking up IDs that didn’t exist. Now if I do that, it’ll return an error message saying no client/assistant/CPA exists with that ID. The submit\_tax\_return() instance method submits a tax return. What makes this different from Tax Assistant’s submit\_tax\_return() is that it will automatically mark the CPA\_check as TRUE and provide a CPA ID. This will change the output of client.check\_return\_status(). The only “weird interaction” that occurs, is that if an assistant already filed a tax return and then a CPA re-files it, then the timestamp gets updated to the newest one. It doesn’t track the original filing timestamp, only the newest one.

TaxAssistant

This Class is honestly really similar to the CPA class. For a second I thought inheritance but the two roles seemed a bit too different in my mind to deal with that. I almost didn’t make this a class, but I realized that if an assistant gains more responsibilities as time goes on, then you could add extra functionalities to reflect those changes. So for now it’s a fairly simple class that can be added upon at a later time. Every method works the same way as the CPA methods, the only difference is that Assistant data will be tracked as opposed to CPA data. The getter class method will take an assistant\_id as a argument and return that assistant’s information and create an object instance. The submit\_tax\_return() does the same here as it did in the CPA class, instead it will store assistant ID and have no CPA ID attached because the CPA hasn’t checked the return yet.