Mahindra École Centrale

CS101 Lab-14
Object Oriented Programming Basics

Aim

This lab will introduce you the basics of Object Oriented Programming (OOP). You will learn to create *classes* and instantiating them to create *objects*. Later you will use object methods for different operations on them.
We will try to develop a simple banking software. After analysing the requirements, it is decided that two basic classes namely *customer* and *account* will be used in the software.

Outcome

At the end of this lab students are expected to be familiar with creating classes and objects, and using their data and function attributes. You will also learn dynamically creating and destroying objects based on the operating environment.

Tasks

Task-1 Creating the customer class

The customer class should have data attributes to store customer name, Dob, address and a list of his accounts. The name, Dob, and address attributes should be assigned within the `__init__` method. The class should also have functions to change the customer address and add or remove accounts.
At this point you can declare an empty dictionary inside the consumer class for storing the accounts. The functions for adding and removing accounts can assume the argument to these functions is a dictionary key. Then use regular dictionary methods to add or remove items. What will be the values, we will see in the next section. For the time being, assign some dummy value. You can use the `del` function or the `pop` method to delete a dictionary item.

Task-2 Creating the account class

Now you can create the account class. The class should have an *account number* and *balance* attributes and should be initialised within the __init__ function. It should accept the *account number* and the *initial deposit* as the initialisation arguments. The class should also have functions to *deposit*, *withdraw* and *calculate interest*. Both the `deposit` and `withdraw` functions should return the present balance in the account as the `return` value and the `calc_interest` function should return the interest received and the balance. The interest rate for the account should be declared as class data attribute (ie. outside all the functions). You can assume that interest paid is simple interest at half yearly based (i.e. n=0.5) on the current balance.

```
1 class account():
2     int_rate = 0.09
3     def __init__(self,accnt_num,init_deposite):
4         .
5         .
```

Now go back to the customer class. **The values of the dictionary used to store the customer accounts are *objects* of *account* class.** It means whenever the function to add an account is invoked, it should create an object of type *account* and add to the dictionary with the key as the account number and the value as the new object created. Since the account class requires initial deposit in its `__init__` function, the add account function should also have it as an argument.

Finally save the file containing these classes as a Python module and open a new file for the next task.

Task-3 Front end

Now let us make the user (operator) interface for our software. This will be your main programme which will be using the classes that we created earlier. For this you have to `import` the module containing the classes that we saved before. At first the interface should show a menu for the operator with the following options.

1. Add a new customer
2. Remove customer
3. Add a new account
4. Remove an account
5. Deposit to an account
6. Withdraw from an account
7. Calculate interest for all customers
8. Change interest rate

Clue: Just print these options on the screen

Now the system should wait for the operator input.

Clue: `input()` with out a message?

After this based on the operator input, the system should take the corresponding actions. It can also check whether any option which is not listed is entered and print an error message.

Clue: if-elif-else ?

For example if the operator selects to add a new customer, system should ask details regarding the customer name, date of birth, address etc. Then create an *object* from the *customer* class and pass these information as the arguments. **Every customer can be stored in a *dictionary*, with the customer name as the *key* and the object as the *value*.**

```
1  .
2  .
3  customers = {}  #Empty dictionary
4  get the customer name and details
5  customers[customer_name] = customer(customer_name,dob,address) #↩
       Creating the object and adding to the dictionary
6  .
7  .
```

When a new account has to be added, the system should ask for the customer name, who will be linked with this account and the initial deposit. Then call the customer object method to add the account. You can use a variable to generate account number, which will be incremented each time after a new account is created. Similarly for removing an account, ask for the customer name, then list all the account numbers linked with this customer to help the operator. **This could be a little bit long instruction such as customers_dictionary[customer].accounts_dictionary.keys()** Now wait for the operator to enter the account number and call the remove account method from the customer object using this account number. Similar logic can be used for depositing and withdrawing from accounts.

When the calculate interest option is selected, interest should be calculated for every account belonging to each customer and printout each account's present balance and interest earned. When you change the interest rate, remember to access it as a class attribute rather than an object attribute.

Finally put your whole code (except the customer dictionary and account number initialisation) inside an infinite loop, so that after a menu item is processed, the system again displays the menu.