TUTORED & PRACTICAL WORK N°05

INNER CLASS, COLLECTIONS & MULTI-INHERITANCE

In Python, it is possible to nest classes (one class inside another). The purpose of nested classes is to group classes that go together, which makes the source code more readable and maintainable.
Some concrete examples of the use of inner classes in the real world:

Exercise 1

1. Create a class `PW_5_1_CPU` that contains:
   - Private attribute `price`;
   - The `display()` method which allows to display the CPU information.
   - The inner class `PW_5_1_PROCESSOR` which contains in its turn:
     - Private attribute `nbCores`;
     - Private attribute `nameManufacturer`;
     - The method `getCache()` which returns the size of the cache memory of the processor (in our case return the value 3).
     - The method `getFrequency()` which returns the clock frequency of the processor (in our case return the value 5).
     - The `display()` method which allows to display the processor information.
   - The inner class `PW_5_1_RAM` which contains in its turn:
     - Private attribute `size`;
     - Private attribute `name_manufacturer`;
     - The `display()` method which allows to display the ram information.
2. In the main program create an object of class `PW_5_1_CPU`, then create an object of the class `PW_5_1_PROCESSEUR` and an object of the class `PW_5_1_RAM`.
3. Display the output of the two methods `getCache()` and `getFrequency()`.
4. Display the information of CPU, processor and ram.

Exercise 2

1. Create the class `PW_5_2_MOTHERBOARD` which contains:
   - The private attribute `model`;
   - The method `display()` which displays the model of the motherboard.
   - The inner class `PW_5_2_USB` which contains in its turn:
     - The private attribute `usb2` initialized to 2;
     - The private attribute `usb3` initialized to 1;
     - The method `display()` which displays the number of `usb2` and `usb3` ports.
2. In the main program, display the model and the number of usb ports of a motherboard.

Exercise 3

1. Create an abstract class `PW_5_3_abs` containing the non-implemented method `display()`.
2. Create the class `PW_5_3_A`.
3. Add in this class the inner class `PW_5_3_A1` which expends the class `TD_5_3_abs`.
4. Create the class `PW_5_3_B` sub-class of `PW_5_3_A`, which contains the method `getObj()` to return an object of the class `PW_5_3_A1`.
5. In the main program, create an object of the class `PW_5_3_B`, get on object class `PW_5_3_A1`, then call its method `display()`;

Exercise 4

We need to create two classes, one for full-time employees `FullTimeEmployee` and the second for hourly employees `HourlyEmployee`. To do that, we will create a model class `Employee` that we will follow to create the two classes.
The common properties (`full_name, last_name`) will be declare in the model class, a model method `get_salary()` which will be defined in the two specific classes. Another model method `display()` will be declared in the model class et defined in the specific classes.
The salary of full-time employees is defined as a property in the `FullTimeEmployee` class, and initialized in the constructor.
The salary of hourly employees is calculated by multiplying the number of hours by the price per hour.

- Define these classes.
- In the main program create an object of full-time employee and another of hourly employee.
- Display their information.

Exercise 5

Create a class `PW_5_5_A` with an inner class `PW_5_5_A1` containing an attribute `'a'` initialized to the value `5`. Create a second class `PW_5_5_B` with an inner class `PW_5_5_B1` that extends the first inner class `PW_5_5_A1`.
In the main program, create an object of class `PW_5_5_B,` and an object of class `PW_5_5_B1,` then print the value of the attribute `'a'` of the second object.

Exercise 6

Consider the following classes:

```
class Address:
    def __init__(self, street, city):
        self.street = str(street)
        self.city = str(city)

    def show(self):
        print(self.street)
        print(self.city)

class Person:
    def __init__(self, name, email):
        self.name = name
```

```
        self.email= email

    def show(self):
        print(self.name + ' ' + self.email)
```

1. Add a new class `Contact` that inherits from both `Address` and `Person`. This class should have a `display()` method.
2. Add a class `Notebook` that is a dictionary that maps people's names to a `Contact` object. This class should have a `display()` method, and `add(self, name, email, street, city)` method.
3. Test the code with the code below:

```
myNotebook = Notebook()
myNotebook.add('Ali', 'ali@hns-re2sd.dz', '5, rue de l'independance',
'Batna')
myNotebook.add('Kader', 'kader@hns-re2sd.dz', '6, rue de l'independance',
'Batna')

myNotebook.display()
```