Object Oriented Programming in Python

Thu Huong Nguyen[1] and **Si Thin Nguyen**[2]

[1]*nthuong@vku.udn.vn*
[2]*nsthin@vku.udn.vn*
[1,2]Faculty of Computer Science, VKU

1 EXERCISES WITH SOLUTIONS

1.1 Write a Python program to create a Person class including private instance attributes: "name" and "age"; a class attribute "count"; a class method "increment_count" (count of how many instances of the class are created), a method "greeting()"; a static method "cls_information()" (return information of class) and a method "str()" (return information of an instance)

[ ]:
```
class Person:
    # class attribute
    count =0
    # constructor
    def __init__(self,name, age):
        self.__name= name
        self.__age = age
        Person.increment_count()
    # class method
    @classmethod
    def increment_count(cls):
        cls.count+=1
    # instance method
    def greeting(self):
        print ("------Hello------")
    # static method
    @staticmethod
    def cls_information():
        print( '='*40+'\nClass infomation: \n- Class name: '+str(Person.
↪__name__)+'\n- Base Classes: '+str(Person.__bases__)+'\n- Number of Person:␣
↪'+str(Person.count)+'\n'+'='*40)
    # special method
    def __str__(self):
        return 'My name is '+self.__name+"\nI am "+str(self.__age)+" years old"
```

1.2 Create two objects from Person class, invoke greeting() method and display information of them via "print()" method. Then, invoking "cls_information" method to display information of class Person

[ ]:
```
a = Person("A",19)
b = Person("B",20)
a.greeting()
print(a)
b.greeting()
print(b)
Person.cls_information()
```

```
------Hello------
My name is A
I am 19 years old
------Hello------
My name is B
I am 20 years old
========================================
Class infomation:
- Class name: Person
- Base Classes: (<class 'object'>,)
- Number of Person: 2
========================================
```

1.3 Create Employee class inheriting to Person class with some supplements: an instance variable – "salary"; an class varibale – "emp_count". Finally, let's overide "cls_information" method, "increment_count" and "str()" method.

[ ]:
```
class Employee(Person):
    # Class variable
    emp_count=0
    # constructor
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.__salary= salary
        Employee.increment_count()
    # class method
    @classmethod
    def increment_count(cls):
        cls.emp_count+=1
    # overriding - static method
    @staticmethod
    def cls_information():
        print( '='*40+'\nClass infomation: \n- Class name: '+str(Employee.
↪__name__)+'\n- Base Classes: '+str(Employee.__bases__)+'\n- Number of Employee:
↪ '+str(Employee.emp_count)+'\n- Number of Person: '+str(Person.
↪count)+'\n'+'='*40)
```

```
    # overriding special method
    def __str__(self):
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nMY ROLE␣
↪IS EMPLOYEE"
```

1.4 Test all information by creating two objects from Employee class

[ ]:
```
c = Employee("C", 21, 600)
d = Employee("D", 20, 650)
c.greeting()
print(c)
d.greeting()
print(d)
Employee.cls_information()
```

```
------Hello------
My name is C
I am 21 years old
My salary is 600
MY ROLE IS EMPLOYEE
------Hello------
My name is D
I am 20 years old
My salary is 650
MY ROLE IS EMPLOYEE
========================================
Class infomation:
- Class name: Employee
- Base Classes: (<class '__main__.Person'>,)
- Number of Employee: 2
- Number of Person: 4
========================================
```

1.5 Create Manager class inheriting to Person class with some supplements: an instance variable – "salary"; an instance variable – "bonus"; an class varibale – "man_count". Finally, let's overide "cls_information" method; "increment_count" method and "str()" method.

[ ]:
```
class Manager(Person):
    # Class variable
    man_count=0
    # constructor
    def __init__(self, name, age, salary, bonus):
        super().__init__(name, age)
        self.__salary= salary
        self.__bonus=bonus
        Manager.increment_count()
```

```
    # class method
    @classmethod
    def increment_count(cls):
        cls.man_count+=1
    # overriding - static method
    @staticmethod
    def cls_information():
        print( '='*40+'\nClass infomation: \n- Class name: '+str(Manager.
↪__name__)+'\n- Base Classes: '+str(Manager.__bases__)+'\n- Number of Managers:␣
↪'+str(Manager.man_count)+'\n- Number of Person: '+str(Person.
↪count)+'\n'+'='*40)
    # overriding special method
    def __str__(self):
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nBonus␣
↪is "+str(self.__bonus)+"\nMY ROLE IS MANAGEMENT"
```

1.6 Test all information by creating two objects from Manager class

[ ]:
```
e = Manager("E", 21, 600,50)
f = Manager("F", 20, 650,50)
e.greeting()
print(e)
f.greeting()
print(f)
Manager.cls_information()
```

```
------Hello------
My name is E
I am 21 years old
My salary is 600
Bonus is 50
MY ROLE IS MANAGEMENT
------Hello------
My name is F
I am 20 years old
My salary is 650
Bonus is 50
MY ROLE IS MANAGEMENT
========================================
Class infomation:
- Class name: Manager
- Base Classes: (<class '__main__.Person'>,)
- Number of Managers: 4
- Number of Person: 8
========================================
```

1.7 Change Person class to abstract class by creating abstract methods: "increment_count", "cls_information", "get_salary()". Then, let's definite these method in Employee class and Manager class

[ ]:
```
from abc import ABC, abstractclassmethod, abstractmethod
class Person(ABC):
    # constructor
    def __init__(self,name, age):
        self.__name= name
        self.__age = age
    # abstract class
    @abstractmethod
    def get_salary(self):
        pass
    def increment_count(cls):
        pass
    # static method
    @staticmethod
    def cls_information():
        pass
    # special method
    def __str__(self):
        return '-'*40+'\nHello'+'\nMy name is '+self.__name+"\nI am "+str(self.
 ↪__age)+" years old"
class Employee(Person):
    # Class variable
    emp_count=0
    # constructor
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.__salary= salary
        Employee.increment_count()
    # class method
    @classmethod
    def increment_count(cls):
        cls.emp_count+=1
    # overrding - get_salary() method
    def get_salary(self):
        return self.__salary
    # overriding - static method
    @staticmethod
    def cls_information():
        print( '='*40+'\nClass infomation: \n- Class name: '+str(Employee.
 ↪__name__)+'\n- Base Classes: '+str(Employee.__bases__)+'\n- Number of Employee:
 ↪ '+str(Employee.emp_count))
    # overriding special method
    def __str__(self):
```

```
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nMy␣
↪total salary is "+str(self.get_salary())+"\nMY ROLE IS EMPLOYEE"
class Manager(Person):
    # Class variable
    man_count=0
    # constructor
    def __init__(self, name, age, salary, bonus):
        super().__init__(name, age)
        self.__salary= salary
        self.__bonus=bonus
        Manager.increment_count()
    # class method
    @classmethod
    def increment_count(cls):
        cls.man_count+=1
    # overrding - get_salary() method
    def get_salary(self):
        return self.__salary+self.__bonus
    # overriding - static method
    @staticmethod
    def cls_information():
        print( '='*30+'\nClass infomation: \n- Class name: '+str(Manager.
↪__name__)+'\n- Base Classes: '+str(Manager.__bases__)+'\n- Number of Managers:␣
↪'+str(Manager.man_count))
    # overriding special method
    def __str__(self):
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nBonus␣
↪is "+str(self.__bonus)+"\nMy total salary is "+str(self.get_salary())+"\nMY␣
↪ROLE IS MANAGEMENT"
```

1.8 Test all information by creating two objects from Employee class and two objects from Manager class

[ ]:
```
# Two objects from Employee class
c = Employee("C", 21, 600)
d = Employee("D", 20, 650)
print(c)
print(d)
Employee.cls_information()
# Two objects from Manager class
e = Manager("E", 21, 600,50)
f = Manager("F", 20, 650,50)
print(e)
print(f)
Manager.cls_information()
```

```
Hello
My name is C
I am 21 years old
My salary is 600
My total salary is 600
MY ROLE IS EMPLOYEE
----------------------------------------
Hello
My name is D
I am 20 years old
My salary is 650
My total salary is 650
MY ROLE IS EMPLOYEE
========================================
Class infomation:
- Class name: Employee
- Base Classes: (<class '__main__.Person'>,)
- Number of Employee: 2
----------------------------------------
Hello
My name is E
I am 21 years old
My salary is 600
Bonus is 50
My total salary is 650
MY ROLE IS MANAGEMENT
----------------------------------------
Hello
My name is F
I am 20 years old
My salary is 650
Bonus is 50
My total salary is 700
MY ROLE IS MANAGEMENT
==============================
Class infomation:
- Class name: Manager
- Base Classes: (<class '__main__.Person'>,)
- Number of Managers: 2
```

1.9 Use "@property" and "@.setter" decoratores for all instance variable of Employee and Person class.

[ ]:
```python
from abc import ABC, abstractclassmethod, abstractmethod
class Person(ABC):
    # constructor
    def __init__(self,name, age):
```

```
        self.__name= name
        self.__age = age
    @property
    def name(self):
        return self.__name
    @name.setter
    def name(self,value):
        self.__name=value
    @property
    def age(self):
        return self.__age
    @age.setter
    def age(self,value):
        self.__age=value
    # abstract class
    @abstractmethod
    def get_salary(self):
        pass
    def increment_count(cls):
        pass
    # static method
    @staticmethod
    def cls_information():
        pass
    # special method
    def __str__(self):
        return '-'*40+'\nHello'+'\nMy name is '+self.__name+"\nI am "+str(self.
 ↪__age)+" years old"
class Employee(Person):
    # Class variable
    emp_count=0
    # constructor
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.__salary= salary
        Employee.increment_count()
    @property
    def salary(self):
        return self.__salary
    @salary.getter
    def salary(self,value):
        self.__salary=value
    # class method
    @classmethod
    def increment_count(cls):
        cls.emp_count+=1
    # overrding - get_salary() method
```

```
    def get_salary(self):
        return self.__salary
    # overriding - static method
    @staticmethod
    def cls_information():
        print( '='*40+'\nClass infomation: \n- Class name: '+str(Employee.
 ↪__name__)+'\n- Base Classes: '+str(Employee.__bases__)+'\n- Number of Employee:
 ↪ '+str(Employee.emp_count))
    # overriding special method
    def __str__(self):
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nMy␣
 ↪total salary is "+str(self.get_salary())+"\nMY ROLE IS EMPLOYEE"

#Employee.cls_information()
class Manager(Person):
    # Class variable
    man_count=0
    # constructor
    def __init__(self, name, age, salary, bonus):
        super().__init__(name, age)
        self.__salary= salary
        self.__bonus=bonus
        Manager.increment_count()
    @property
    def salary(self):
        return self.__salary
    @salary.setter
    def salary(self,value):
        self.__salary=value
    @property
    def bonus(self):
        return self.__bonus
    @bonus.setter
    def bonus(self,value):
        self.__bonus=value
    # class method
    @classmethod
    def increment_count(cls):
        cls.man_count+=1
    # overrding - get_salary() method
    def get_salary(self):
        return self.__salary+self.__bonus
    # overriding - static method
    @staticmethod
    def cls_information():
```

```
        print( '='*30+'\nClass infomation: \n- Class name: '+str(Manager.
↪__name__)+'\n- Base Classes: '+str(Manager.__bases__)+'\n- Number of Managers:␣
↪'+str(Manager.man_count))
    # overriding special method
    def __str__(self):
        return super().__str__()+"\nMy salary is "+str(self.__salary)+"\nBonus␣
↪is "+str(self.__bonus)+"\nMy total salary is "+str(e.get_salary())+"\nMY ROLE␣
↪IS MANAGEMENT"
```

1.10 Test all information

[ ]:
```
#Test information for Employee
c = Employee("C", 21, 600)
print(c)
# Change information
c.__name= 'Nguyen Van C'
c.__age=20
c.__salary=650
print('-'*40+'\nMY UPDATE INFORMATION\n'+'-'*40)
print('-'*40+'\nHello'+'\nMy name is "'+c.__name+'"\nI am "'+str(c.__age)+'"␣
↪years old'+'\nMy salary is "'+str(c.__salary)+'"\nMy total salary is "'+str(c.
↪__salary)+'"\nMY ROLE IS EMPLOYEE')
# Test information for Manager
e = Manager("E", 21, 600,50)
print(e)
e.__name="Tran Thi E"
e.__age=20
e.__salary=650
e.__bonus=60
#Test information
print('-'*40+'\nMY UPDATE INFORMATION\n'+'-'*40)
print('-'*40+'\nHello'+'\nMy name is "'+e.__name+'"'+'\nI am "'+str(e.__age)+'"␣
↪years old'+'\nMy salary is "'+str(e.__salary)+'\nBonus is "'+str(e.
↪__bonus)+'\nMy total salary is "'+str(e.__salary+e.__bonus)+'"\nMY ROLE IS␣
↪MANAGEMENT')
```

```
----------------------------------------
Hello
My name is C
I am 21 years old
My salary is 600
My total salary is 600
MY ROLE IS EMPLOYEE
----------------------------------------
MY UPDATE INFORMATION
----------------------------------------
----------------------------------------
Hello
```

```
My name is "Nguyen Van C"
I am "20" years old
My salary is "650"
My total salary is "650"
MY ROLE IS EMPLOYEE
----------------------------------------
Hello
My name is E
I am 21 years old
My salary is 600
Bonus is 50
My total salary is 650
MY ROLE IS MANAGEMENT
----------------------------------------
MY UPDATE INFORMATION
----------------------------------------
----------------------------------------
Hello
My name is "Tran Thi E"
I am "20" years old
My salary is "650
Bonus is "60
My total salary is "710"
MY ROLE IS MANAGEMENT
```

2 DO IT YOURSELF

2.1 Create a Python class called BankAccount which represents a bank account, having private attributes: ID (interger type, automatically increasing from 1 value), name (name of the account), balance.

- Use **@property** and **@.setter** decoratores for all attributes
- Create a **Deposit()** method which manages the deposit actions.
- Create a **Withdrawal()** method which manages withdrawals actions.
- Create a **str()** method to display account details.

2.2 Write a Python class Shape with two abstract method "square" and "perimeter". Creating two class inheriting from Shape class and declaring these methods

2.3 Write a program to illustrate *bubble sort* algorithm, using terminology of OOP paradigm. Design a program that allows users to input the number of array. Generate random integer in number range input. Display unsorted array and sorted array using bubble sort.

2.4 Write a program to manage information of student. The program implements terminology of OOP paradigm. A student information consists of *ID, Student Name, Semester, Course Name*. The program allows use to create list of student, update/delete student information. On the other hand, use can search student(s) and sort result by student name. Main Screen as below:

WELCOME TO STUDENT MANAGEMENT

1. Create
2. Find and Sort
3. Update/Delete
4. Report
5. Exit

2.5 Write a Python class Teacher inherited from abstract class Person to manage information of teachers with some supplementing attributes: *TeachingClasses*, *TeachingHours*, *SalaryPerHour* and some overiding methods: *"increment_count"*, "cls_information", "*get_salary()*" in which salary is computed as follows: Salary=SalaryPerHour * TeachingHours

2.6 Write a Python program used in *Product Management System* including

1. Class **Product** with the attributes: ***Name***, ***Description***, ***Price***, ***ListRating*** (list of scores evaluated by the customers to the products) and the method ***viewInfo()*** (print name, price and description of the products)
2. Class **Shop** with the attribute ***ListProduct*** (the list of products) and the methods: ***addProduct()*** (requesting the users to enter the information of a new product and then adding it into ***ListProduct***), ***removeProduct()*** (requesting the user to enter the name of the product and then find and remove it from ***ListProduct***), ***viewProductList()*** (print information (name, price, rating)) of the products in ***ListProduct*** by calling ***viewInfo()*** of class *Product* and the rating for each product will be as the average of the scores from ***ListRating***)
3. A menu:
   PRODUCT MANAGEMENT SYSTEM
   - Add new product
   - Remove product
   - View product list
   - Search product

2.7 Write a Python program used for a *Library Management System* described as follows:

- Books are put on the shelves (Each shelf contains a list of books).
- Bookshelves are classified on different fields, e.g. Information Technology, Mathematics, Literature, Languages, Novel, History,.... and are numbered incrementally starting with 1.
- Each book has its own code (ISBN), title, author, publisher, publishing year, price.
- Books are arranged on shelves according to their field and alphabetical ordering (A–>Z).
- The management system can retrieve the most expensive and cheapest books according to each bookshelf by entering the ordering number of the shelf.
- In order to find books, the users can enter the code and/or the name of the book and/or the name of the author. The system also can return information of one or more books corresponding to the above criteria.