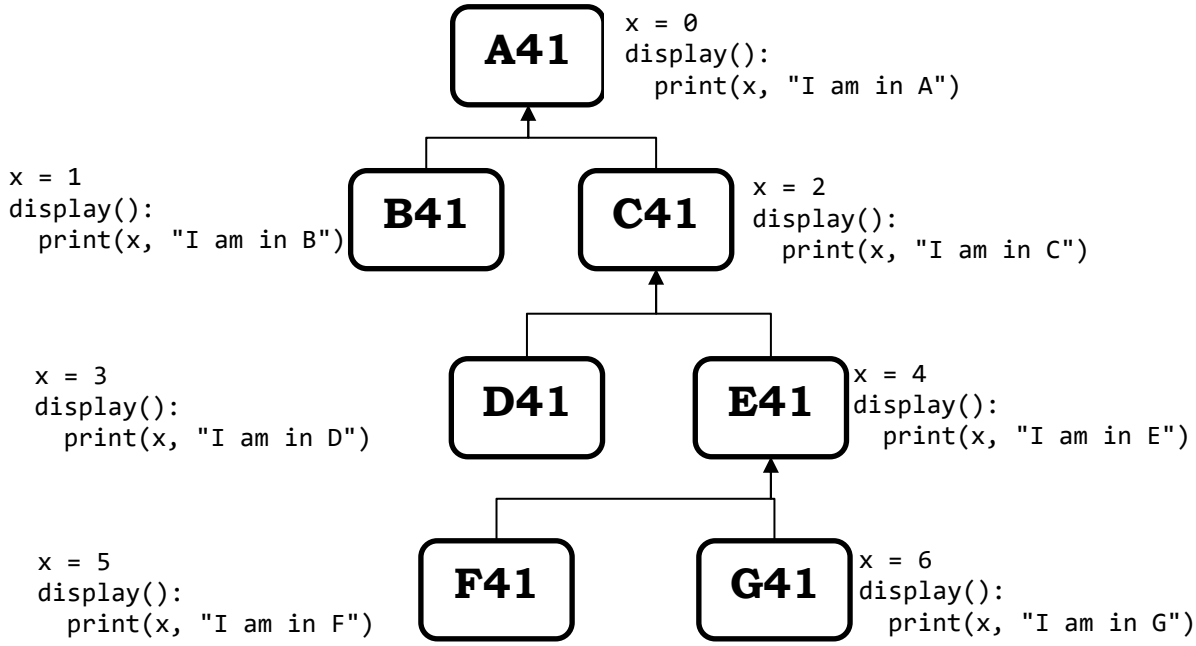
TUTORED & PRACTICAL WORK N°04

INHERITANCE AND POLYMORPHISM

Exercise 1

Consider the following class diagram:



1. What is the OOP concept used in the definition of `diplay()` method?
2. Develop all these classes, and give the output of each instruction:

```
a = A41()
b = B41()
c = C41()
d = D41()
e = E41()
f = F41()
g = G41()
a.display()
b.display()
a.display()

c.display()
a.display()

d.display()
c.display()

e.display()
c.display()
```

```
f.display()
e.display()

g.display()
e.display()
a.display()
```

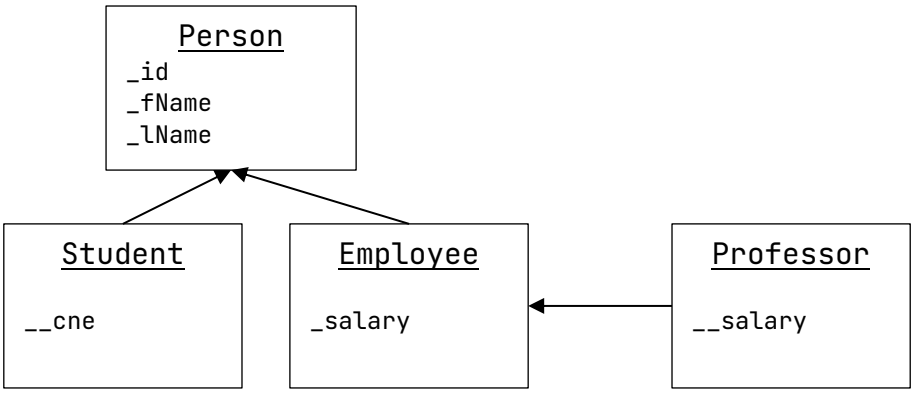
Exercise 2

Give the output of the following program:

```
from abc import ABC, abstractmethod
class I(ABC):
    @abstractmethod
    def x(self):
        pass
    @abstractmethod
    def y(self):
        pass

class A42(I):
    def w(self):
        print("I am in A42.w")
    def x(self):
        print("I am in A42.x")
    def y(self):
        print("I am in A42.y")

class TW_4_2(A42):
    def y(self):
        print("I am in TW_4_2.y")
    def z(self):
        self.w()
        self.x()
        super(TW_4_2, self).y()
        #A42.y(self)

aa = A42()
bb = TW_4_2()
bb.z()
bb.y()
```

Exercise 3

Consider the following class diagram:



1. Develop these classes.
    - Each class must contain a constructor.
    - Each class must redefine the public method **`toString()`** of the class `Person`. This method returns a string containing the information of the class.
2. In the main program, create:
    - Two students;
    - Two employees;
    - Two professors;
    - And finally, display the information of each person.
3. What is the utility of the `'salary'` attribute of the class `Professor`? i.e. Why do we define `salary` attribute in `Professor` class, while it inherits it from the `employee` class?

Exercise 4

1. Give the output of the following program:

```
class A(object):
    def __init__(self):
        super(A, self).__init__()
        print("I am in A")

class B(object):
    def __init__(self):
        super(B, self).__init__()
        print("I am in B")

class C(A, B):
    def __init__(self):
        super(C, self).__init__()
        print("I am in C")

print(C.mro())
c = C()
```

2. Give the output of the previous program if we omit the line:

```
        super(A, self).__init__()
```

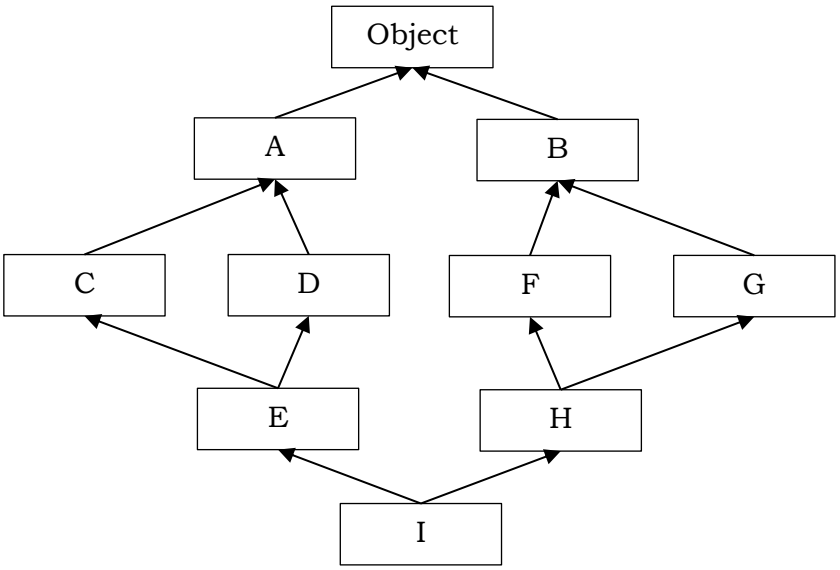
3. Give the output of the following program:

```
class A(object):
    def __init__(self):
        print("I am in A")
        super(A, self).__init__()
```

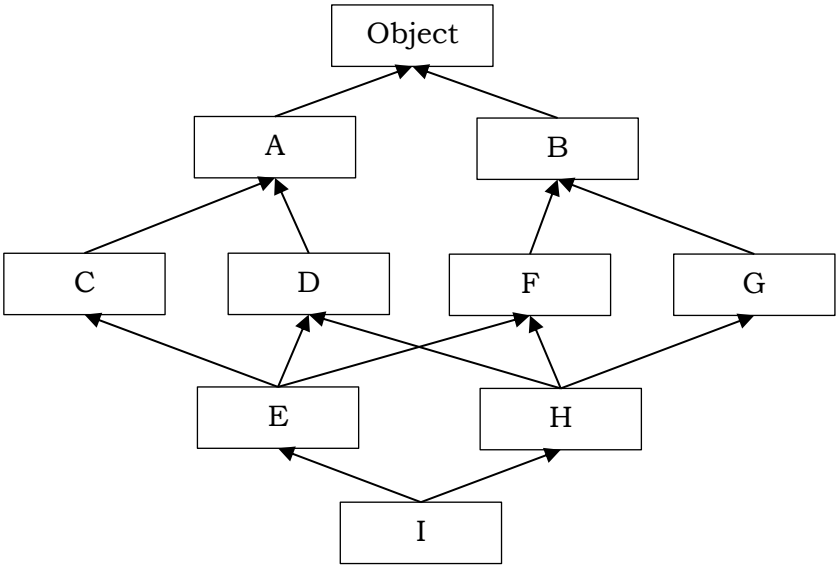
```
class B(object):
    def __init__(self):
        print("I am in B")
        super(B, self).__init__()

class C(A, B):
    def __init__(self):
        print("I am in C")
        super(C, self).__init__()

print(C.mro())
c = C()
```

Exercise 5

1. Give the MRO of the following class diagram:



2. Consider the following class diagram. Give the MRO of class 'I':

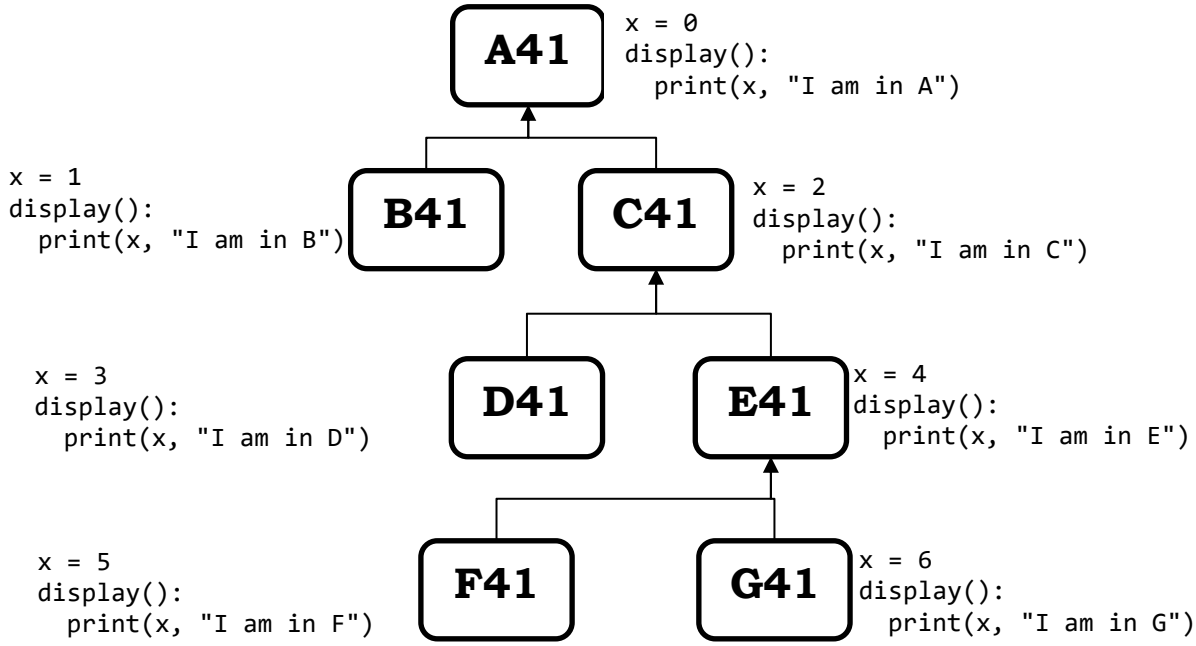


Correction of TW/PW n°04

INHERITANCE AND POLYMORPHISM

Exercise 1

Consider the following class diagram:



3. What is the OOP concept used in the definition of `diplay()` method?
4. Develop all these classes, and give the output of each instruction:

```
a = A41()
b = B41()
c = C41()
d = D41()
e = E41()
f = F41()
g = G41()
a.display()
b.display()
a.display()

c.display()
a.display()

d.display()
c.display()

e.display()
c.display()

f.display()
e.display()
```

```
g.display()
e.display()
a.display()
```

Solution

1. To define `display()` method, we used the '**overriding**' polymorphism type.
2. Definition of classes:

```
class A41:
    def __init__(self):
        self.x = 0
    def display(self):
        print(self.x, " I am in A")
class B41(A41):
    def __init__(self):
        self.x = 1
    def display(self):
        print(self.x, " I am in B")
class C41(A41):
    def __init__(self):
        self.x = 2
    def display(self):
        print(self.x, " I am in C")
class D41(C41):
    def __init__(self):
        self.x = 3
    def display(self):
        print(self.x, " I am in D")
class E41(C41):
    def __init__(self):
        self.x = 4
    def display(self):
        print(self.x, " I am in E")
class F41(E41):
    def __init__(self):
        self.x = 5
    def display(self):
        print(self.x, " I am in F")
class G41(E41):
    def __init__(self):
        self.x = 6
    def display(self):
        print(self.x, " I am in G")
```

Output:

```
0  I am in A
1  I am in B
0  I am in A
2  I am in C
0  I am in A
3  I am in D
2  I am in C
```

```
4  I am in E
2  I am in C
5  I am in F
4  I am in E
6  I am in G
4  I am in E
0  I am in A
```

Exercise 2

Give the output of the following program:

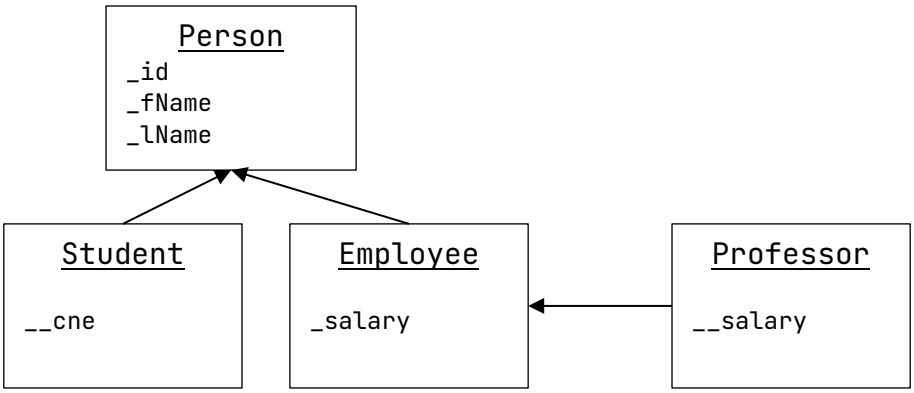
```
from abc import ABC, abstractmethod
class I(ABC):
    @abstractmethod
    def x(self):
        pass
    @abstractmethod
    def y(self):
        pass

class A42(I):
    def w(self):
        print("I am in A42.w")
    def x(self):
        print("I am in A42.x")
    def y(self):
        print("I am in A42.y")

class TW_4_2(A42):
    def y(self):
        print("I am in TW_4_2.y")
    def z(self):
        self.w()
        self.x()
        super(TW_4_2, self).y()
        #A42.y(self)

aa = A42()
bb = TW_4_2()
bb.z()
bb.y()
```

Solution

```
I am in A42.w
I am in A42.x
I am in A42.y
I am in TW_4_2.y
```

Exercise 3

Consider the following class diagram:



1. Develop these classes.
    - Each class must contain a constructor.
    - Each class must redefine the public method **`toString()`** of the class `Person`. This method returns a string containing the information of the class.
2. In the main program, create:
    - Two students;
    - Two employees;
    - Two professors;
    - And finally, display the information of each person.
3. What is the utility of the `'salary'` attribute of the class `Professor`? i.e. Why do we define `salary` attribute in `Professor` class, while it inherits it from the `employee` class?

Solution

```
class Person:
    def __init__(self, i, f, l):
        self._id = i
        self._fName = f
        self._lName = l
    def toString(self):
        ret = 'Identifier: ' + str(self._id) + '\nFirst name: ' +
self._fName + '\nLast name: ' + self._lName
        return ret

class Student(Person):
    def __init__(self, i, f, l, c):
        super(Student, self).__init__(i, f, l)
        self.__cne = c
    def toString(self):
        ret = '*** Student ***' + '\nCNE: ' + str(self.__cne) + '\n' +
super(Student, self).toString() + '\n'
        return ret

class Employee(Person):
    def __init__(self, i, f, l, s):
        super(Employee, self).__init__(i, f, l)
        self._salary = s
    def toString(self):
        ret = '*** Employee ***' + '\n' + super(Employee, self).toString()
```

```
+ '\nSalary: ' + str(self._salary) + '\n'
        return ret

class Professor(Employee):
    def __init__(self, i, f, l, s):
        super(Professor, self).__init__(i, f, l, s)
        self.__salary = s
    def toString(self):
        ret = '*** Professor ***' + '\n' + super(Professor,
self).toString() + '\n'
        return ret

std1 = Student(1, 'FS1', 'LS1', 11111)
std2 = Student(2, 'FS2', 'LS2', 22222)

emp1 = Employee(1, 'FE1', 'LE1', 50000)
emp2 = Employee(2, 'FE2', 'LE2', 60000)

prof1 = Professor(1, 'FP1', 'LP1', 70000)
prof2 = Professor(2, 'FP2', 'LP2', 80000)

print(std1.toString())
print(std2.toString())
print(emp1.toString())
print(emp2.toString())
print(prof1.toString())
print(prof2.toString())
```

Display:

```
*** Student ***
CNE: 11111
Identifier: 1
First name: FS1
Last name: LS1

*** Student ***
CNE: 22222
Identifier: 2
First name: FS2
Last name: LS2

*** Employee ***
Identifier: 1
First name: FE1
Last name: LE1
Salary: 50000

*** Employee ***
Identifier: 2
First name: FE2
Last name: LE2
Salary: 60000
```

```
*** Professor ***
*** Employee ***
Identifier: 1
First name: FP1
Last name: LP1
Salary: 70000


*** Professor ***
*** Employee ***
Identifier: 2
First name: FP2
Last name: LP2
Salary: 80000
```

The attribute `'salary'` of the class `'Professor'` allows to consider this attribute as `private` whereas it is defined protected in the super class.

Exercise 4

1. Give the output of the following program:

```
class A(object):
    def __init__(self):
        super(A, self).__init__()
        print("I am in A")

class B(object):
    def __init__(self):
        super(B, self).__init__()
        print("I am in B")

class C(A, B):
    def __init__(self):
        super(C, self).__init__()
        print("I am in C")

print(C.mro())
c = C()
```

2. Give the output of the previous program if we omit the line:

```
        super(A, self).__init__()
```

3. Give the output of the following program:

```
class A(object):
    def __init__(self):
        print("I am in A")
        super(A, self).__init__()

class B(object):
    def __init__(self):
        print("I am in B")
        super(B, self).__init__()

class C(A, B):
```

```
    def __init__(self):
        print("I am in C")
        super(C, self).__init__()

print(C.mro())
c = C()
```

Solution

1. The display is:

```
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>]
I am in B
I am in A
I am in C
```

2. The display is:

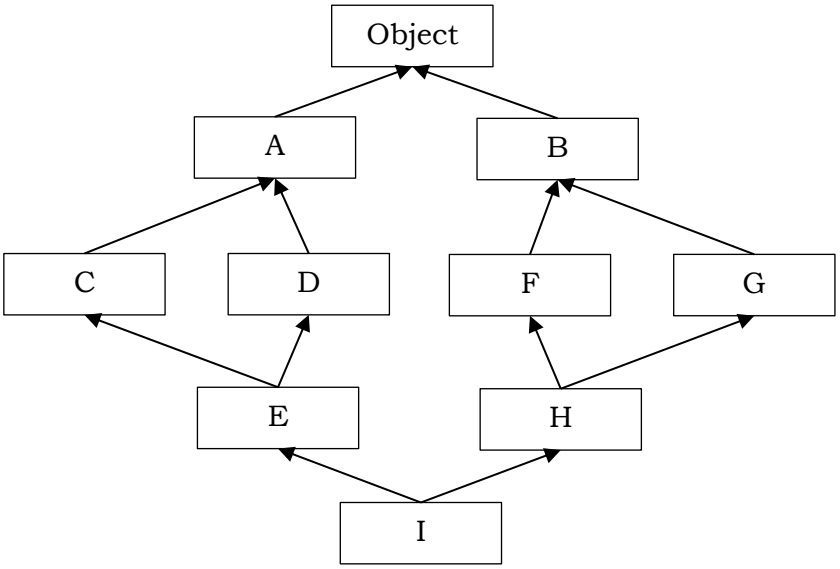
```
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>]
I am in A
I am in C
```

3. The display is:

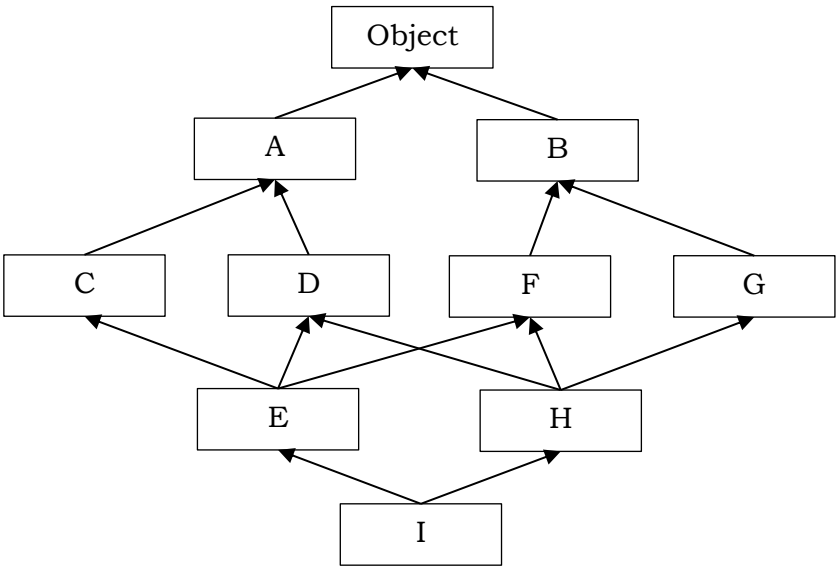
```
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>]
I am in C
I am in A
I am in B
```

Exercise 5

1. Give the MRO of the following class diagram:



2. Consider the following class diagram. Give the MRO of class 'I':



Solution

1. The MRO of class 'I' is:

```
I E C D A H F G B object
```

2. The MRO of class 'I' is:

```
I E C H D A F G B object
```