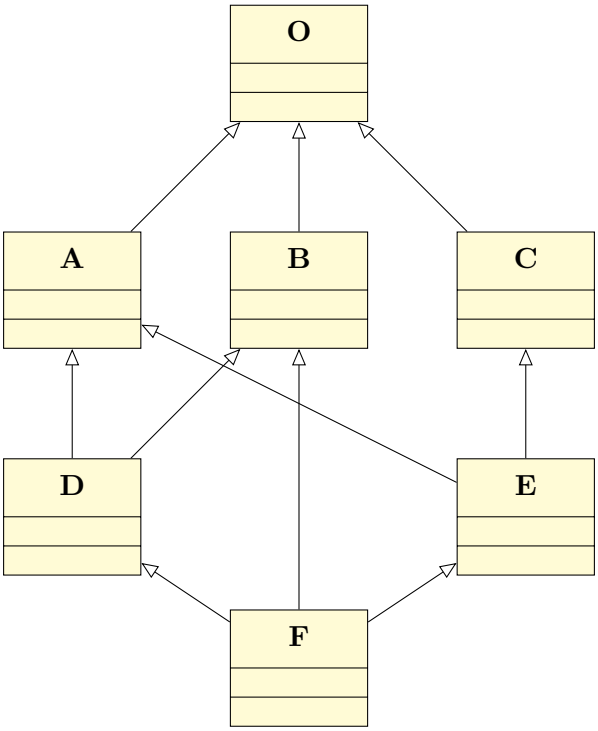
Tutorial 4 : Object Oriented Programming

Khoi Dang Do - 1711807
khoi.do.1711807@hcmut.edu.vn

September 2019

Question 1

Given the following class schema, (left to right order)



(a) Apply method resolution order of Python 3 to find the class search path of each class in the above class schema? If there are errors in the class search path, please change the order of super classes of some class to make all the class search path success.

L(O) = [O]; L(A) = [A, O]; L(B) = [B, O]; L(C) = [C, O]

L(D) = D + merge(L(A), L(B), [A, B]) = [D, A, B, O]

L(E) = E + merge(L(A), L(C), [A, C]) = [E, C, A, O]

L(F) = F + merge(L(D), L(B), L(E), [D, B, E]) (invalid)

L(F) = F + merge(L(D), L(E), L(B), [D, E, B]) = [F, D, E, C, A, B, O] (valid)

(b) If x contains an object of F, which will method foo be called by x.foo() using your successful class schema? foo method of A class will be called.

Question 2

(a) Write class Rational using Python.

(b) Make sure that when creating a Rational object without any argument (Rational()), object Rational whose n is 0 and d is 1 is created.

(c) Rewrite method + so that it can accepts parameter **that** in type **int**. Make sure that the new code calls recursively to the old code.

```
class Rational :
    # greatest common divisor
    def __gcd(self, n, d):
        if(d == 0):
            return n
        else:
            return self.__gcd(d, n%d)


    # constructor / instance attributes
    def __init__(self, numer=0, denom=1):
        if denom == 0 :
            raise SystemExit("denom must not be zero!")
        else:
            common = self.__gcd(numer, denom)
            self.numer = numer//common
            self.denom = denom//common
    # override add operator (rational)
    def __add__(self, other):
        if isinstance(other, int):
            return self + Rational(other)
        else:
            common = self.__gcd(self.numer, self.denom)
            new_numer = self.numer*other.denom + other.numer*self.denom
            new_denom = self.denom*other.denom
            return Rational(new_numer, new_denom)
    # override string convert method
    def __str__(self):
        return str(self.numer) + "/" + str(self.denom)
```

Question 3

Redefine the example on Case class using Python

(a) Write method print for class **Number** to print the value of field **num** in **Number**.

(b) Make an object that represents the expression **(x + 0.2) * 3** and assign it to variable t.

(c) Write method eval that can evaluate an expression return a **Number** object. The operators which may be appeared in an expression are "+", "-", "*", "/". Assume that the value of all variables is 1. For example, t.eval().print() => 3.6

```
from abc import ABC

class Expr(ABC):
    pass

class Var(Expr):
    def __init__(self, name = "", *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.name = name

class Number(Expr):
    def __init__(self, num=0.0, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.num = num
    def print(self):
        print(self.num) # print number attribute

class BinOp(Expr):
    def __init__(self, operator, left, right, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.operator = operator
        self.left = left
        self.right = right

    def eval(self):
        if self.operator == "+":
            return Number(self.left + self.right)
        elif self.operator == "-":
            return Number(self.left - self.right)
        elif self.operator == "*":
            return Number(self.left * self.right)
        elif self.operator == "/":
            return Number(self.left / self.right)
```