**Харківський національний економічний університет**

**імені Семена Кузнеця**

**ЗВІТ**

**З ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 10**

**за дисципліною: *“*Технології програмування**”

**на тему: “Принципи ООП”**

**Варіант: 4**

**Виконав: студент факультету Інформаційних технологій**

**2 курсу, спец. Кібербезпека,**

**групи 6.04.125.010.21.2**

**Бойко Вадим Віталійович**

**Перевірив:**

**Міхєєв Іван Андрійович**

**ХНЕУ ім. С. Кузнеця**

**2022**

**Мета:** Метою роботи є набуття навичок створення програм на основі технологій об’єктно-орієнтованого програмування: створення класів, визначення механізму спадкування

**Завданя:** Написати програму з використанням визначення класів для збереження та виводу інформації відповідно до варіанту завдання. Реалізувати ієрархію класів із застосуванням механізму спадкування. Продемонструвати роботу із об’єктами кожного створеного класу

****

Я написав наступне:

Клас деталь

*class* Detal:  
 *def \_\_init\_\_*(self, *name* = ''):  
 self.name = *name* self.work = *False  
  
 def* do(self):  
 self.work = *not* self.work  
 print(f'{self.name} works: {self.work}')  
  
 *def* isWorking(self):  
 *return* self.work

Клас механізм, який наслідує деталь та є поліморфізм

*class* Mechanism(Detal):  
 *def \_\_init\_\_*(self, *name* = ''):  
 super().\_\_init\_\_(*name*)  
 self.rotate = *False  
  
 def* action(self):  
 self.rotate = *not* self.rotate  
 print(f'{self.name} rotate: {self.work}')  
  
 *def* do(self):  
 self.work = *not* self.work  
 print(f'{self.name} works: {self.work}')  
 self.action()

Клас виріб, який реалізує механізм залежності

*class* Virib:  
 *def \_\_init\_\_*(self, *name* = '', *mechanisms* = []):  
 self.name = *name* self.work = *False* self.mechanisms = *mechanisms  
  
 def* toggle(self):  
 *if not* self.work:  
 print(f'{self.name} starts work')  
 *for* mechanism *in* self.mechanisms:  
 mechanism.do()  
 print(f'{self.name} is working')  
 *else*:  
 print(f'{self.name} stops')  
 *for* mechanism *in* self.mechanisms:  
 mechanism.do()  
 print(f'{self.name} has been stopped')

Клас вузол, який реалізує механізм залежності, поліморфізм, наслідування, агрегація

*class* Vuzol(Virib):  
 *def \_\_init\_\_*(self, *name* = '', *mechanisms* = [], *safeMechanism* = Mechanism()):  
 super().\_\_init\_\_(*name*, *mechanisms*)  
 self.safeMechanism = *safeMechanism  
   
 def* toggle(self, *number* = 1):  
 *if number* % 2 == 0:  
 *if not* self.safeMechanism.isWorking():  
 self.safeMechanism.do()  
 print(f'{self.name} starts work')  
 *for* mechanism *in* self.mechanisms:  
 mechanism.do()  
 print(f'{self.name} is working')  
 *else*:  
 print('not safe')

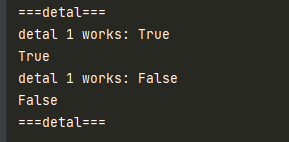
Приклади роботи

Ініціалізація об’єктів

d1 = Detal('detal 1')  
d2 = Detal('detal 2')  
d3 = Detal('detal 3')  
  
m1 = Mechanism('mechanism 1')  
m2 = Mechanism('mechanism 1')  
m3 = Mechanism('mechanism 3')  
  
virib1 = Virib('Virib 1', [d1, m1])  
virib2= Virib('Virib 2', [d2, m2])  
  
safeMechanism = Mechanism('Safe mechanism')  
  
vuzol1 = Vuzol('Vuzol 1', [d3, m3], safeMechanism)

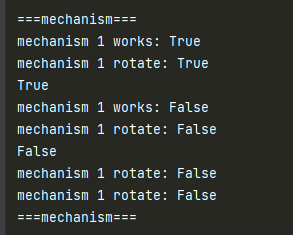
Робота з деталлю

print('===detal===')  
d1.do()  
print(d1.isWorking())  
d1.do()  
print(d1.isWorking())  
print('===detal===')  
print()



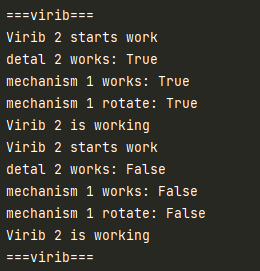
Робота з механізмом

print('===mechanism===')  
m1.do()  
print(m1.isWorking())  
m1.do()  
print(m1.isWorking())  
m1.action()  
m1.action()  
print('===mechanism===')  
print()



Робота з виробом

print('===virib===')  
virib2.toggle()  
virib2.toggle()  
print('===virib===')  
print()



Робота з вузлом

print('===vuzol===')  
vuzol1.toggle(9)  
vuzol1.toggle(10)  
vuzol1.toggle(10)  
print('===vuzol===')  
print()

