**Задание на лабораторную работу №4**

**Тема**: Классы

* Написать скрипт, содержащий в себе все возможности языка, связанные с объектно-ориентированным программированием:
* -Создание класса и -объекта;
* -Инкапсуляция и -статические методы;
* -Наследование и **композиция**1;
* Полиморфизм: -перегрузка операторов и -методов, -абстрактные и -виртуальные методы.
* Проверить работу программы.

Ход работы:  
from abc import ABCMeta, abstractmethod  
  
# Определение класса R для использования в композиции  
class R:  
 def \_\_init\_\_(self):  
 print("r")  
  
# Определение класса Re, использующего композицию с классом R  
class Re:  
 def \_\_init\_\_(self):  
 self.\_rer = R()  
  
# Определение базового класса Stick  
class Stick:  
 def \_\_init\_\_(self, owner, length):  
 self.\_owner = owner # Инкапсуляция: приватное поле \_owner  
 self.\_length = length  
  
 # Переопределение метода для информативного вывода  
 def \_\_str\_\_(self):  
 return f"Owner: {self.\_owner}, Length: {self.\_length}"  
  
 # Статический метод, не требующий доступа к экземпляру  
 @staticmethod  
 def who\_are\_you():  
 print("stick")  
  
 # Метод доступа к приватному полю \_owner через декораторы  
 @property  
 def owner(self):  
 return self.\_owner  
  
 @owner.setter  
 def owner(self, owner):  
 self.\_owner = owner  
  
# Определение класса Hand, наследующего от класса Stick  
class Hand(Stick):  
 def \_\_init\_\_(self, power, \*args):  
 super().\_\_init\_\_(\*args) # Вызов конструктора родительского класса  
 self.\_power = power  
  
 # Переопределение метода для информативного вывода  
 def \_\_str\_\_(self):  
 return super().\_\_str\_\_() + f", Power: {self.\_power}"  
  
 # Полиморфный метод  
 @staticmethod  
 def who\_are\_you():  
 print("hand")  
  
 # Перегрузка оператора +  
 def \_\_add\_\_(self, other):  
 return self.\_power + other.\_power  
  
# Определение абстрактного класса Class1 с абстрактным методом test  
class Class1(metaclass=ABCMeta):  
 @staticmethod  
 @abstractmethod  
 def test(x):  
 pass  
  
# Определение класса Class2, наследующего от Class1, с виртуальным методом test  
class Class2(Class1):  
 @staticmethod  
 def test(x):  
 print(x)  
  
# Основная программа  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Создание и использование объектов класса Stick  
 st = Stick("Maiki", 100)  
 st.owner = "Maiki Maikovich"  
 print(st.owner)  
 print(st)  
 Stick.who\_are\_you()  
  
 print()  
  
 # Создание и использование объектов класса Hand  
 Ken\_hand = Hand(33, 'Ken', 47)  
 Takemichi\_hand = Hand(200, 'Takemichi', 110)  
 print(Ken\_hand)  
 print(Takemichi\_hand)  
 Hand.who\_are\_you()  
  
 print()  
  
 # Полиморфное использование оператора +  
 Maiki\_power = Ken\_hand + Takemichi\_hand  
 print(Maiki\_power)  
  
 # Создание и использование объектов других классов  
 stick = Stick(["Maiki", "Ken", "Takemichi"], [1, 2, 3])  
 print(stick)  
  
 print()  
 # Использование виртуального метода  
 Class2.test("----------------------")  
 # Использование класса, использующего композицию  
 Re()

Результат работы:  
