Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине: «Языки программирования»

Вариант №11

Выполнил:

Студент 2-го курса

Группы ПО-7

Лобан К.Ю.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** Ознакомиться с принципом инкапсуляции, ознакомиться с использованием принципа наследования в Python.

**Ход работы:**

**Задание 1**

# Постановка задачи:

1. Определить пользовательский класс – «Улица»

2. Определить счетчик

3. Определить в классе конструкторы с параметрами и без. Конструктор должен выводить сообщение о количестве объектов.

4. Определить в классе внешние компоненты-функции для получения и установки полей данных.

5. Написать демонстрационную программу, в которой объекты пользовательского класса создаются с помощью неявного использования конструкторов без параметров.

6. Показать в программе явное использование конструкторов с параметрами.

**Код программы:**

class Street:  
 count = 0  
 # Если переменная начинается с двух нижних подчеркиваний, то эта переменная будет доступна только внутри класса  
 # Это инкапсуляция (сокрытие данных)  
  
 # Полиморфизм - способность функции обрабатывать данные разных типов.  
 \_\_street\_name = str()  
 \_\_city = str()  
 \_\_number = int()  
  
 def \_\_init\_\_(self):  
 Street.count += 1  
 print(Street.count)  
  
 def \_\_init\_\_(self, sn="", ct="", nm=0):  
 Street.count += 1  
 print(Street.count)  
  
 print("Конструктор с параметрами вызван")  
 try:  
 self.\_\_street\_name = str(sn) # \_\_street\_name - поле класса  
 self.\_\_city = str(ct)  
 self.\_\_number = int(nm)  
 except ValueError as e:  
 print(e)  
  
 def set\_street\_name(self, sn):  
 try:  
 self.\_\_street\_name = str(sn)  
 except ValueError as e:  
 print(e)  
  
 def set\_city(self, ct):  
 try:  
 self.\_\_city = str(ct)  
 except ValueError as e:  
 print(e)  
  
 def set\_number(self, nm):  
 try:  
 self.\_\_number = int(nm)  
 except ValueError as e:  
 print(e)  
  
 def get\_street\_name(self):  
 return self.\_\_street\_name  
  
 def get\_city(self):  
 return self.\_\_city  
  
 def get\_number(self):  
 return self.\_\_number  
  
 def read(self):  
 try:  
 self.\_\_street\_name = input("Введите название улицы :\n")  
 self.\_\_city = input("Введите название города:\n")  
 self.\_\_number = int(input("Введите номер улицы:\n"))  
 except ValueError:  
 print("Ошибка ввода")  
  
 def show(self):  
 print("Название улицы:" + self.\_\_street\_name + " Название города:" + self.\_\_city + " Номер улицы:" + str(self.\_\_number))  
  
  
# Специальная переменная \_\_name\_\_ будет равна \_\_main\_\_ только, когда запускается как отдельный файл  
# Равна имени скрипта, который импортировал этот скрипт  
if \_\_name\_\_ == "\_\_main\_\_":  
 y = Street() # Создание экземпляра класса. Вызывается конструктор без параметров  
 y1 = Street()  
 y.read()  
 y1.read()  
 y.show()  
 y1.show()  
 x = Street("Советская", "Брест", 19)  
 x.show()  
 c = Street()  
 c.set\_street\_name("Московская")  
 c.set\_city("Брест")  
 c.set\_number(17)  
 c.show()  
 print("Количество улиц:", Street.count)

**Задание 2**

# Постановка задачи:

1. Построить модель предметной области в соответствии со своим вариантом (см. ниже)

2. Для каждого класса создать конструктор и деструктор, выдающий сообщение о своей работе

3. Для каждого класса создать внешние функции установки и получения полей данных

4. Для каждого класса разработать функции, позволяющие представить на экране значения полей данных

5. Для каждого класса разработать функции, позволяющие вводить с консоли значения полей данных

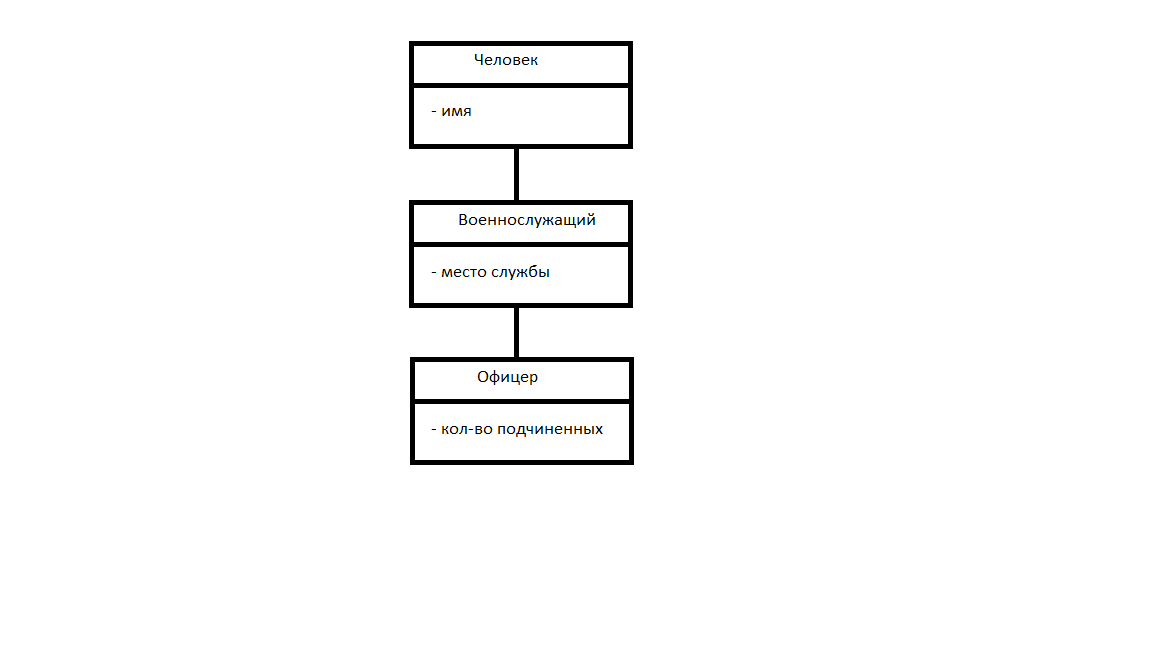
6. Написать демонстрационную программу, иллюстрирующую поочередный вызов конструкторов и деструкторов базового и производного классов

7. Построить диаграмму классов

**Код программы:**

class Human:  
 name = str()  
  
 def \_\_init\_\_(self, nm=""):  
 print("Вызван конструктор Human", self)  
 if type(nm) == str:  
 self.\_\_name = nm  
 else:  
 print("Ошибка ввода")  
  
 def set\_name(self, nm):  
 if type(nm) == str:  
 self.\_\_name = nm  
 else:  
 print("Ошибка ввода")  
  
 def get\_name(self):  
 return self.\_\_name  
  
 def \_\_del\_\_(self):  
 print("Вызван деструктор класса Human ", self)  
  
 def read(self):  
 self.\_\_name = input("Введите имя человека:\n")  
  
 def show(self):  
 print("Имя человека:", self.\_\_name)  
  
  
class Serviceman(Human):  
 \_\_place\_of\_service = str()  
  
 def \_\_init\_\_(self, ps="", nm=""):  
 print("Вызван конструктор класса Serviceman", self)  
 try:  
 self.\_\_place\_of\_service = ps  
 self.\_\_name = nm  
 except ValueError:  
 print("Ошибка ввода")  
  
 def set\_place\_of\_service(self, ps):  
 self.\_\_place\_of\_service = ps  
  
 def get\_place\_of\_service(self):  
 return self.\_\_place\_of\_service  
  
 def \_\_del\_\_(self):  
 print("Вызван деструктор класса Serviceman", self)  
  
 def read(self):  
 self.\_\_name = input("Введите имя военнослужащего:\n")  
 self.\_\_place\_of\_service = input("Введите место службы:\n")  
  
 def show(self):  
 print("Имя человека: ", self.\_\_name)  
 print("Место службы: ", self.\_\_place\_of\_service)  
  
  
class Officer(Serviceman):  
 \_\_number\_of\_subordinates = int()  
  
 def \_\_init\_\_(self, ps="", nm="", ns=0):  
 print("Вызван конструктор класса Officer", self)  
 try:  
 self.\_\_name = nm  
 self.\_\_place\_of\_service = ps  
 self.\_\_number\_of\_subordinates = int(ns)  
 except ValueError:  
 print("Ошибка ввода")  
  
 def \_\_del\_\_(self):  
 print("Вызван деструктор класса Officer", self)  
  
 def set\_number\_of\_subordinates(self, ns):  
 try:  
 self.\_\_number\_of\_subordinates = ns  
 except ValueError:  
 print("Ошибка ввода")  
  
 def get\_number\_of\_subordinates(self):  
 return self.\_\_number\_of\_subordinates  
  
 def read(self):  
 try:  
 self.\_\_name = input("Введите имя военнослужащего:\n")  
 self.\_\_place\_of\_service = input("Введите место службы:\n")  
 self.\_\_number\_of\_subordinates = int(input("Введите количество подчиненных:\n"))  
 except ValueError:  
 print("Ошибка ввода")  
  
 def show(self):  
 print("Имя человека: ", self.\_\_name)  
 print("Место службы: ", self.\_\_place\_of\_service)  
 print("Количество подчиненных:", self.\_\_number\_of\_subordinates)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("Example 1")  
 human1 = Human()  
 human1.set\_name("Олешик Максим")  
 human1.show()  
 print("\n")  
  
 print("Example 2")  
 human2 = Human("Теркин Леонид")  
 human2.show()  
 print("\n")  
  
 print("Example 3")  
 serviceman1 = Serviceman()  
 serviceman1.read()  
 serviceman1.show()  
 print("\n")  
  
 print("Example 4")  
 serviceman2 = Serviceman("Петров Константин", "д. Нижние холмы")  
 serviceman2.show()  
 print("\n")  
  
 print("Example 5")  
 officer1 = Officer()  
 officer1.read()  
 officer1.show()  
 print("\n")  
  
 print("Example 6")  
 officer2 = Officer("Тернов Андрей", "г. Минск", 36)  
 officer2.show()  
 print("\n")

**Диаграмма классов:**



**Вывод:** я ознакомился с принципом инкапсуляции, ознакомился с использованием принципа наследования в Python.