Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный университет”
Кафедра ИИТ

Лабораторная работа №6

По дисциплине “Языки программирования”

Вариант №7

Выполнил:
Кравцевич Г.А. (ПО-7,2)
Проверил:
Дряпко. А. В.

Дата выполнения:
17.09.21

Брест 2021

Цель:
Ознакомиться со способами создания классов и принципами ООП в Python

Задание 1:
Для своего варианта (см.ниже) выполнить следующее:

1. Определить пользовательский класс в соответствии с вариантом задания.
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.

Предметная область:
Отдел кадров

Код программы:
```
class HRDepatrment:
    count = 0
    __employees = list()

    def __init__(self, employees=[]):
        HRDepatrment.count += 1
        print('Count:', HRDepatrment.count)

        self.__employees = employees

    def get_employees(self):
        return self.__employees

    def set_employees(self, employees):
        if len(employees) > 0:
```

```
                self.__employees = employees
        else:
                print('Invalid argument')

    def clear_employees_list(self):
        self.__employees = []


# Usage
if __name__ == '__main__':
    first = HRDepatrment()
    last = HRDepatrment(['Olga', 'George'])

    print(last.get_employees())

    first.set_employees(['Bess', 'Tom'])
    print(first.get_employees())

    first.clear_employees_list()
    last.clear_employees_list()
```

Задание 2:
Для своего варианта выполнить следующее:

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

Предметная область:
Учитель-Ученик-Персона

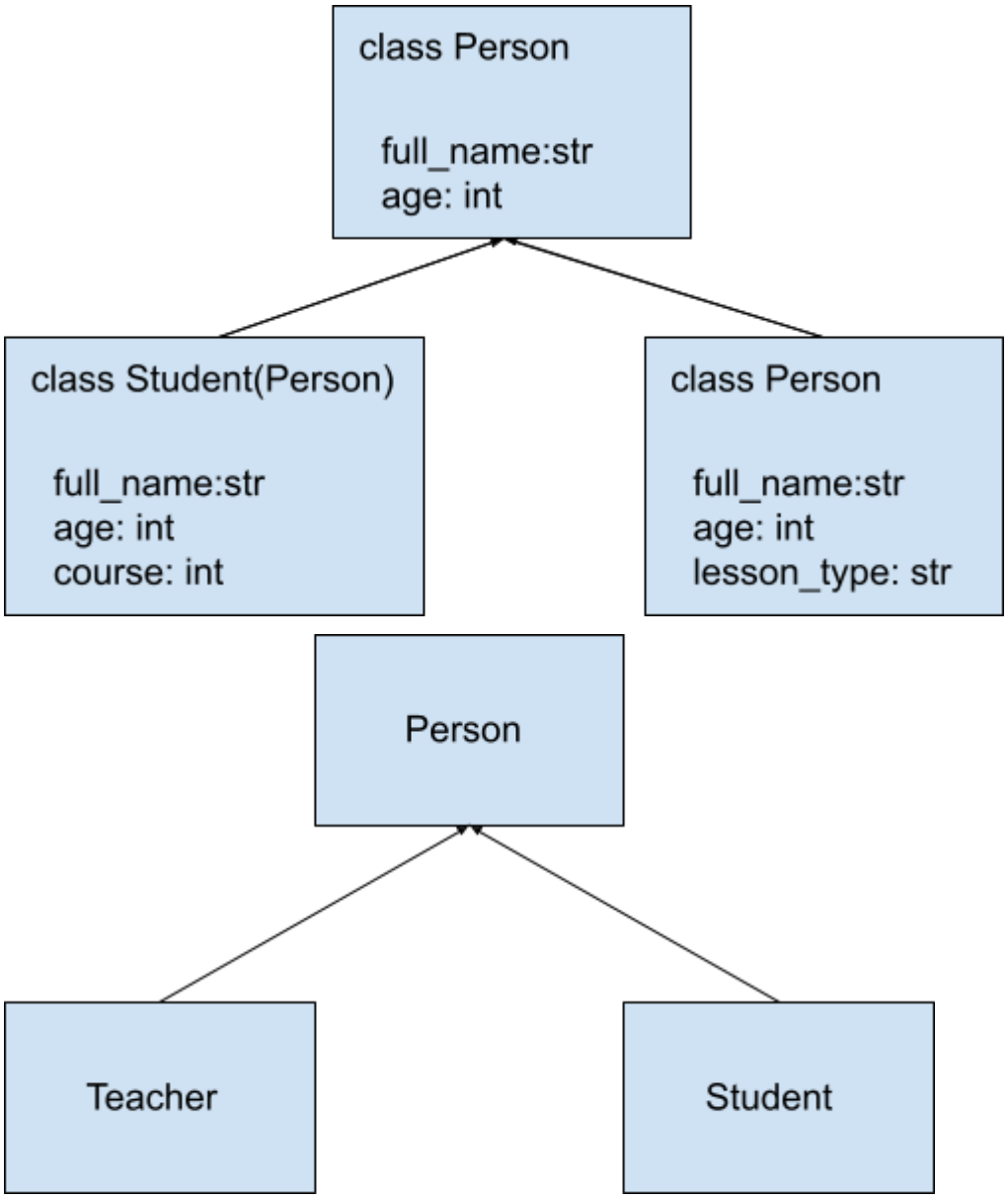
Код программы:
```
class Person:
  __full_name = str()
  __age = int()

  def __init__(self, full_name=None, age=0):
    print('Вызван конструктор класса Person')
    self.__full_name = full_name
    self.__age = age

  def __del__(self):
    print('Вызван деструктор класса Person')

  def set_age(self, age):
    age = int(age)
    if age < 0:
      print('Invalid argument')
      return

    self.__age = age

  def get_age(self):
    return self.__age

  def set_full_name(self, full_name):
    if len(full_name) == 0:
      print('Invalid argument')
      return

    self.__full_name = full_name

  def get_full_name(self):
    return self.__full_name
```

```
  def __str__(self):
    return f'Full name: {self.get_full_name()}\nAge: {self.get_age()}'

  def console_init(self):
    self.set_full_name(input('Full name: '))
    self.set_age(input('Age: '))


class Student(Person):
  __course = int()

  def __init__(self, full_name=None, age=0, course=1):
    print('Вызван конструктор класса Student')
    super().__init__(full_name, age)
    self.__course = course

  def __del__(self):
    print('Вызван деструктор класса Student')

  def set_course(self, course):
    if 5 < course < 1:
      print('Invalid argument')
      return

    self.__course = course

  def get_course(self):
    return self.__course

  def __str__(self):
    return f'Full name: {self.get_full_name()}\nAge: {self.get_age()}\nCourse:
{self.get_course()}'

  def console_init(self):
    super(Student, self).console_init()
    self.set_course(input('Course: '))
```

```
class Teacher(Person):
  __lesson_type = str()

  def __init__(self, full_name=None, age=0, lesson_type=None):
    print('Вызван конструктор класса Teacher')
    super().__init__(full_name, age)
    self.__lesson_type = lesson_type

  def __del__(self):
    print('Вызван деструктор класса Teacher')

  def set_lesson_type(self, lesson_type):
    if len(lesson_type) == 0:
      print('Invalid argument')

    self.__lesson_type = lesson_type

  def get_lesson_type(self):
    return self.__lesson_type

  def __str__(self):
    return f'Full name: {self.get_full_name()}\nAge: {self.get_age()}\nLesson
type: {self.get_lesson_type()}'

  def console_init(self):
    super(Teacher, self).console_init()
    self.set_lesson_type(input('Lesson type: '))


if __name__ == '__main__':
  person = Person('Tom', 12)
  student = Student('Bob', 18, 1)
  teacher = Teacher('Mr. Jon', 40, 'Math')

  print(str(person))
  print(str(student))
```

```
print(str(teacher))

teacher.console_init()
print(teacher)
```

Иерархия классов:



Вывод:

Ознакомился с принципом наследования в Python