Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине: «ЯП»

Тема: «Классы. Инкапсуляция»

Выполнил:

Студент 2 курса

Группы ПО-7(2)

Мозоль И.П.

Проверил:

Бойко Д.О.

Брест 2021

Цель работы: Ознакомиться с принципом инкапсуляции и с использованием принципа наследования в Python.

Вариант 1

Задание:

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.

**Класс: Выпускник ВУЗа.**

**Код программы:**

class Graduate:  
 counter = 0  
  
 def \_\_init\_\_(self, name="Igor", faculty="FEIS", year\_release=2024):  
 Graduate.counter += 1  
 print(f"Now we have {Graduate.counter} people")  
 self.\_\_name = name  
 self.\_\_faculty = faculty  
 self.\_\_year\_release = year\_release  
 print(f'{self.counter} constructor')  
  
 @property  
 def name(self) -> str:  
 return self.\_\_name  
  
 @name.setter  
 def name(self, name) -> None:  
 self.\_\_name = name  
  
 @property  
 def faculty(self) -> str:  
 return self.\_\_faculty  
  
 @faculty.setter  
 def faculty(self, faculty) -> None:  
 self.\_\_faculty = faculty  
  
 @property  
 def year(self) -> int:  
 return self.\_\_year\_release  
  
 @year.setter  
 def year(self, year) -> None:  
 if year.isdigit():  
 self.\_\_year\_release = year  
 else:  
 print("Wrong number")  
  
 def show(self) -> None:  
 print(f"Name - {self.\_\_name}, Faculty name - {self.\_\_faculty}, Release year - {self.\_\_year\_release}")  
  
 def read(self) -> None:  
 self.\_\_name = input("Input Name - ")  
 self.\_\_faculty = input("Input your faculty name - ")  
 try:  
 self.\_\_year\_release = int(input("Input your release year -"))  
 except ValueError:  
 print("Bro, u need write number, not string.")  
  
  
class Person:  
 def \_\_init\_\_(self, name: str = "Igor", age: int = 18, sex: str = "Man"):  
 self.\_name = name  
 self.\_age = age  
 self.\_sex = sex  
 print(f"Person constructor for {type(self).\_\_name\_\_} worked")  
  
 @property  
 def name(self) -> str:  
 return self.\_name  
  
 @name.setter  
 def name(self, name: str) -> None:  
 self.\_name = name  
  
 @property  
 def age(self) -> int:  
 return self.\_age  
  
 @age.setter  
 def age(self, age: int) -> None:  
 if age.isdigit():  
 self.\_age = age  
 else:  
 print("Wrong number")  
  
 @property  
 def sex(self) -> str:  
 return self.\_sex  
  
 @sex.setter  
 def sex(self, sex: str) -> None:  
 self.\_sex = sex  
  
 def show(self) -> None:  
 print(f"Name - {self.\_name}")  
 print(f"Age - {self.\_age}")  
 print(f"Sex - {self.\_sex}")  
  
 def \_\_del\_\_(self):  
 print(f"Destructor for {type(self).\_\_name\_\_} worked")  
  
 def console\_set(self):  
 self.\_name = input("Input your name -")  
 try:  
 self.\_age = int(input("Input your age -"))  
 except ValueError:  
 print("Bro, u need write number, not string.")  
 self.\_sex = input("Input your sex -")  
  
  
class Student(Person):  
 def \_\_init\_\_(self, name: str = "Igor", age: int = 18, sex: str = "Man", course: int = 2):  
 super().\_\_init\_\_(name, age, sex)  
 self.\_course = course  
 print(f"Student constructor for {type(self).\_\_name\_\_} worked")  
  
 @property  
 def course(self) -> int:  
 return self.\_course  
  
 @course.setter  
 def course(self, course: int) -> None:  
 self.\_course = course  
  
 def show(self) -> None:  
 super().show()  
 print(f"Course number - {self.\_course}")  
  
 def \_\_del\_\_(self):  
 print(f"Destructor for {type(self).\_\_name\_\_} worked")  
  
 def console\_set(self):  
 super().console\_set()  
 try:  
 self.\_course = int(input("Input your course number -"))  
 except ValueError:  
 print("Bro, u need write number, not string.")  
  
  
class SportsMan(Student):  
 def \_\_init\_\_(self, name: str = "Igor", age: int = 18, sex: str = "Man", course: int = 2, medals\_amount: int = 5):  
 super().\_\_init\_\_(name, age, sex, course)  
 self.\_medals\_amount = medals\_amount  
 print(f"Sportsman constructor for {type(self).\_\_name\_\_} worked")  
  
 @property  
 def medals(self) -> int:  
 return self.\_medals\_amount  
  
 @medals.setter  
 def medals(self, medals: int) -> None:  
 self.\_medals\_amount = medals  
  
 def show(self) -> None:  
 super().show()  
 print(f"Medals amount - {self.\_medals\_amount}")  
  
 def \_\_del\_\_(self):  
 print(f"Destructor for {type(self).\_\_name\_\_} worked")  
  
 def console\_set(self):  
 super().console\_set()  
 try:  
 self.\_medals\_amount = int(input("Input your medals amount -"))  
 except ValueError:  
 print("Bro, u need write number, not string.")  
  
  
def main():  
 input("Task 1. Press Enter")  
 graduate\_1 = Graduate()  
 graduate\_1.show()  
 graduate\_1.year = input("Input ur release year -")  
 graduate\_1.show()  
 graduate\_1.read()  
 graduate\_1.show()  
 graduate\_2 = Graduate("Aloha", "FISE")  
 graduate\_2.show()  
 input("Task 2. Press Enter")  
 person = Person()  
 person.show()  
 student = Student("Boris", 25, "Man")  
 print(student.age)  
 sportsman = SportsMan("Anna", 50, "Woman", 2, 1000)  
 sportsman.console\_set()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод: При выполнении данной лабораторной работы я изучил такие принципы ООП как наследование, инкапсуляция и полиморфизм