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)

Панкратов Р.C.

Проверил:

Бойко Д.О.

2021

Вариант 3

Цель работы: Ознакомиться с принципом инкапсуляции. Ознакомиться с конструкторами и деструкторами в Python. Ознакомиться с указателем self. Инкапсуляция в Python. Ознакомиться с использованием принципа наследования 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.

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

Код программы:

from typing import Optional  
  
  
class InstanceCountMixin:  
 \_\_instance\_counts = 0  
  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 InstanceCountMixin.\_\_instance\_counts += 1  
 print(f"Create {cls.\_\_name\_\_}")  
 return super().\_\_new\_\_(cls)  
  
 @property  
 def instance\_counts(self) -> int:  
 return self.\_\_instance\_counts  
  
 def \_\_del\_\_(self):  
 print(f"Del {self.\_\_class\_\_.\_\_name\_\_}")  
 InstanceCountMixin.\_\_instance\_counts -= 1  
  
  
class ReprMixin:  
 def \_\_repr\_\_(self) -> str:  
 patched\_mro\_classes = list(self.\_\_class\_\_.\_\_mro\_\_)  
 patched\_mro\_classes.remove(InstanceCountMixin)  
 allowed\_classes = set(mro\_class.\_\_name\_\_ for mro\_class in patched\_mro\_classes)  
 fields\_to\_print = set(field for field in dir(self) if field.split("\_\_")[0][1:] in allowed\_classes)  
  
 return "{}({})".format(  
 self.\_\_class\_\_.\_\_name\_\_,  
 ", ".join(f"{field.split('\_\_')[-1]}={getattr(self, field)}" for field in fields\_to\_print)  
 )  
  
  
class PartyMember(InstanceCountMixin, ReprMixin):  
 def \_\_init\_\_(self, age: Optional[int] = None, name: Optional[str] = None, party\_name: Optional[str] = None):  
 self.\_\_age = age  
 self.\_\_party\_name = party\_name  
 self.\_\_name = name  
  
 @property  
 def age(self) -> Optional[int]:  
 return self.\_\_age  
  
 @age.setter  
 def age(self, value: int) -> None:  
 self.\_\_age = value  
  
 @property  
 def party\_name(self) -> Optional[str]:  
 return self.\_\_party\_name  
  
 @party\_name.setter  
 def party\_name(self, value: str) -> None:  
 self.\_\_party\_name = value  
  
 @property  
 def name(self) -> Optional[str]:  
 return self.\_\_name  
  
 @name.setter  
 def name(self, value: str) -> None:  
 self.\_\_name = value  
  
  
class SchoolBoy(InstanceCountMixin, ReprMixin):  
 def \_\_init\_\_(self, age: Optional[int] = None, name: Optional[str] = None, height: Optional[float] = None):  
 self.\_\_age = age  
 self.\_\_name = name  
 self.\_\_height = height  
  
 @property  
 def age(self) -> Optional[int]:  
 return self.\_\_age  
  
 @age.setter  
 def age(self, value: int) -> None:  
 self.\_\_age = value  
  
 @property  
 def name(self) -> Optional[str]:  
 return self.\_\_name  
  
 @name.setter  
 def name(self, value: str) -> None:  
 self.\_\_name = value  
  
 @property  
 def height(self) -> Optional[int]:  
 return self.\_\_height  
  
 @height.setter  
 def height(self, value: int) -> None:  
 self.\_\_height = value  
  
  
class Person(SchoolBoy):  
 def \_\_init\_\_(  
 self,  
 age: Optional[int] = None,  
 name: Optional[str] = None,  
 height: Optional[float] = None,  
 is\_education: bool = False,  
 earnings: Optional[float] = None  
 ) -> None:  
 super().\_\_init\_\_(age, name, height)  
 self.\_\_is\_education = is\_education  
 self.\_\_earnings = earnings  
  
 @property  
 def is\_education(self) -> bool:  
 return self.\_\_is\_education  
  
 @is\_education.setter  
 def is\_education(self, value: bool) -> None:  
 self.\_\_is\_education = value  
  
 @property  
 def earnings(self) -> Optional[float]:  
 return self.\_\_earnings  
  
 @earnings.setter  
 def earnings(self, value: float) -> None:  
 self.\_\_earnings = value  
  
  
class Teacher(Person):  
 def \_\_init\_\_(  
 self,  
 age: Optional[int] = None,  
 name: Optional[str] = None,  
 height: Optional[float] = None,  
 earnings: Optional[float] = None,  
 focus: Optional[str] = None,  
 status: Optional[str] = None  
 ) -> None:  
 super().\_\_init\_\_(age, name, height, True, earnings)  
 self.\_\_focus = focus  
 self.\_\_status = status  
  
 @property  
 def focus(self) -> Optional[str]:  
 return self.\_\_focus  
  
 @focus.setter  
 def focus(self, value: str) -> None:  
 self.\_\_focus = value  
  
 @property  
 def status(self) -> Optional[str]:  
 return self.\_\_status  
  
 @status.setter  
 def status(self, value: str) -> None:  
 self.\_\_status = value  
  
  
def task\_1():  
 print("Task 1")  
 age = int(input("Enter age: "))  
 name = str(input("Enter name: "))  
 party\_name = str(input("Enter party\_name: "))  
  
 party\_member = PartyMember()  
 party\_member.age = age  
 party\_member.name = name  
 party\_member.party\_name = party\_name  
 print(party\_member)  
  
 party\_member\_with\_constructor = PartyMember(age, name, party\_name)  
 print(party\_member\_with\_constructor)  
  
 print(f"Instance counts: {party\_member\_with\_constructor.instance\_counts}")  
  
  
def task\_2():  
 print("Task 2")  
 age = int(input("Enter age: "))  
 name = str(input("Enter name: "))  
 height = float(input("Enter height: "))  
  
 school\_boy = SchoolBoy()  
 school\_boy.age = age  
 school\_boy.name = name  
 school\_boy.height = height  
 print(school\_boy)  
  
 school\_boy\_2 = SchoolBoy(age, name, height)  
 print(school\_boy\_2)  
  
 is\_education = bool(input("Enter is education: "))  
 earnings = float(input("Enter earnings: "))  
  
 person = Person()  
 person.age = age  
 person.name = name  
 person.height = height  
 person.is\_education = is\_education  
 person.earnings = earnings  
 print(person)  
  
 person\_2 = Person(age, name, height, is\_education, earnings)  
 print(person\_2)  
  
 focus = str(input("Enter focus: "))  
 status = str(input("Enter status: "))  
  
 teacher = Teacher()  
 teacher.age = age  
 teacher.name = name  
 teacher.height = height  
 teacher.earnings = earnings  
 teacher.focus = focus  
 teacher.status = status  
 print(teacher)  
  
 teacher\_2 = Teacher(age, name, height, earnings, focus, status)  
 print(teacher\_2)  
  
  
def main():  
 task\_1()  
 task\_2()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод: В ходе выполнения данной работы я научился работать с классами, познакомился с такими терминами как инкупсуляция, наследование.