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)

Угляница И.Н

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:

Задание 1:

**class** Apartment:  
 count = 0  
  
 **def** \_\_init\_\_(self, n\_square=20, n\_price=300, n\_city=**"Brest"**):  
 Apartment.count += 1  
 self.\_\_square = n\_square  
 self.\_\_price = n\_price  
 self.\_\_city = n\_city  
 print(**f"Constructor worked {**Apartment.count**} time"**)  
  
 @property  
 **def** square(self) -> int:  
 **return** self.\_\_square  
  
 @square.setter  
 **def** square(self, n\_square) -> **None**:  
 self.\_\_square = n\_square  
  
 @property  
 **def** price(self) -> int:  
 **return** self.\_\_price  
  
 @price.setter  
 **def** price(self, n\_price) -> **None**:  
 self.\_\_price = n\_price  
  
 @property  
 **def** city(self) -> str:  
 **return** self.\_\_city  
  
 @city.setter  
 **def** city(self, n\_city) -> **None**:  
 self.\_\_city = n\_city  
  
 **def** Show(self) -> **None**:  
 print(**f"Square = {**self.\_\_square**},Price = {**self.\_\_price**},City = {**self.\_\_city**}"**)  
  
  
**def** main():  
 flat1 = Apartment()  
 flat2 = Apartment(21, 450, **"Minsk"**)  
 flat1.Show()  
 flat2.Show()  
 flat1.square = 777  
 print(flat1.square)  
 flat1.Show()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2:

**class** Learner:  
 **def** \_\_init\_\_(self, n\_name=**"Ivan"**, n\_birth=**"06.07.2003"**):  
 self.\_name = n\_name  
 self.\_birth = n\_birth  
 print(**"Constructor worked for Learner"**)  
  
 **def** \_\_del\_\_(self):  
 print(**f"destructor worked{**self.\_name**}"**)  
  
 @property  
 **def** name(self) -> str:  
 **return** self.\_name  
  
 @name.setter  
 **def** name(self, n\_name) -> **None**:  
 self.\_name = n\_name  
  
 @property  
 **def** birth(self) -> str:  
 **return** self.\_birth  
  
 @birth.setter  
 **def** birth(self, n\_birth) -> **None**:  
 self.\_birth = n\_birth  
  
  
**class** Student(Learner):  
 **def** \_\_init\_\_(self, n\_name=**"Ivan"**, n\_birth=**"06.07.2003"**, n\_faculty=**"FEIS"**):  
 super().\_\_init\_\_(n\_name, n\_birth)  
 self.\_\_faculty = n\_faculty  
 print(**"Constructor worked for Student"**)  
  
 **def** \_\_del\_\_(self):  
 print(**f"Destructor worked for class student"**)  
  
 **def** show(self) -> **None**:  
 print(**f"Student: {**self.\_name**} {**self.\_birth**} faculty: {**self.\_\_faculty**}"**)  
  
 **def** input\_data(self) -> **None**:  
 self.\_name = input(**"Input student name "**)  
 self.\_birth = input(**"Input student birthday "**)  
 self.\_\_faculty = input(**"Input student faculty "**)  
  
 @property  
 **def** faculty(self) -> str:  
 **return** self.\_\_faculty  
  
 @faculty.setter  
 **def** faculty(self, n\_faculty) -> **None**:  
 self.\_\_faculty = n\_faculty  
  
  
**class** Undergraduate(Learner):  
 **def** \_\_init\_\_(self, n\_name=**"Ivan"**, n\_birth=**"06.07.2003"**, n\_direction=**""**):  
 super(Undergraduate, self).\_\_init\_\_(n\_name, n\_birth)  
 self.\_\_direction = n\_direction  
 print(**"Constructor worked for undergraduate"**)  
  
 **def** \_\_del\_\_(self):  
 print(**"Destructor worked for undergraduate"**)  
  
 **def** show(self) -> **None**:  
 print(  
 **f"Undergraduate: {**self.\_name**} {**self.\_birth**} direction: {**self.\_\_direction**}"** )  
  
 **def** input\_data(self) -> **None**:  
 self.\_name = input(**"Input student name "**)  
 self.\_birth = input(**"Input student birthday "**)  
 self.\_\_direction = input(**"Input student direction "**)  
  
 @property  
 **def** direction(self) -> str:  
 **return** self.\_\_direction  
  
 @direction.setter  
 **def** direction(self, n\_direction) -> **None**:  
 self.\_\_direction = n\_direction  
  
  
**def** main():  
 s1 = Student()  
 s1.input\_data()  
 s1.show()  
 u1 = Undergraduate()  
 u1.birth = **"777"** u1.show()  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

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