МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Информатика»
Тема: Парадигмы программирования

Студентка гр. 0382 _____ Здобнова К.Д.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Изучить принципы объектно-ориентированного программирования на языке python.

Задание.

Система классов для градостроительной компании

Базовый класс -- схема дома HouseScheme:

class HouseScheme:

Поля объекта класса HouseScheme:

- количество жилых комнат
- площадь (в квадратных метрах, не может быть отрицательной)
- совмещенный санузел (значениями могут быть или False, или True)

При создании экземпляра класса HouseScheme необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'

Дом деревенский CountryHouse:

class CountryHouse: # Класс должен наследоваться от HouseScheme

Поля объекта класса CountryHouse:

- количество жилых комнат
- жилая площадь (в квадратных метрах)
- совмещенный санузел (значениями могут быть или False, или True)
- количество этажей
- площадь участка

При создании экземпляра класса CountryHouse необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'

Метод __str__()

Преобразование к строке вида:

'Country House: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный

санузел>, Количество этажей <количество этажей>, Площадь участка <площадь участка>.

Метод __eq__()'

Метод возвращает True, если два объекта класса равны и False иначе.

Два объекта типа CountryHouse равны, если равны жилая площадь, площадь участка, при этом количество этажей не отличается больше, чем на 1. '"

Квартира городская Apartment:

class Apartment: # Класс должен наследоваться от HouseScheme

Поля объекта класса Apartment:

- количество жилых комнат
- площадь (в квадратных метрах)
- совмещенный санузел (значениями могут быть или False, или True)
- этаж (может быть число от 1 до 15)
- куда выходят окна (значением может быть одна из строк: N, S, W, E)

При создании экземпляра класса Apartment необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'

Метод __str__()

Преобразование к строке вида:

'Apartment: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Этаж <этаж>, Окна выходят на <куда выходят окна>.'

Переопределите список list для работы с домами:

Деревня:

class CountryHouseList: # список деревенских домов -- "деревня", наследуется от класса list

Конструктор:

1. Вызвать конструктор базового класса

2. Передать в конструктор строку name и присвоить её полю name созданного объекта'"

Метод append(p_object):

Переопределение метода append() списка.

В случае, если p_object - деревенский дом, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: 'Invalid type <тип_объекта p_object>'

Метод total_square():

Посчитать общую жилую площадь:

Жилой комплекс:

class ApartmentList: # список городских квартир -- ЖК, наследуется от класса list

Конструктор:

1. Вызвать конструктор базового класса

2. Передать в конструктор строку name и присвоить её полю name созданного объекта'"

Метод extend(iterable):

Переопределение метода extend() списка.

В случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.

Метод floor_view(floors, directions):

В качестве параметров метод получает диапазон возможных этажей в виде списка (например, [1, 5]) и список направлений из ('N', 'S', 'W', 'E').

Метод должен выводить квартиры, этаж которых входит в переданный диапазон (для [1, 5] это 1, 2, 3, 4, 5) и окна которых выходят в одном из переданных направлений. Формат вывода:

<Направление_1>: <этаж_1>

<Направление_2>: <этаж_2>

...

Направления и этажи могут повторятся. Для реализации используйте функцию filter().

Основные теоретические положения.

Парадигма программирования - это совокупность принципов, методов и понятий, которые определяют методику реализации программ.

Лямбда-выражения - это специальный элемент синтаксиса для создания анонимных (т.е. без имени) функций по месту их использования. Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций.

Синтаксис:

lambda аргумент1, аргумент2,..., аргументN : выражение

Функция filter(<функция>, <объект>) возвращает объект-итератор, состоящий из тех элементов итерируемого объекта <объект>, для которых <функция> является истиной. Функция <функция> применяется для каждого элемента итерируемого объекта <объект>.

Для функции filter(<функция>, <объект>) в качестве аргумента <функция> может быть передано lambda-выражение.

Выполнение работы.

Класс *HouseScheme().* Поля объекта: *number_of_rooms* – количество жилых комнат, *square* – площадь, *combined_bathroom* – совмещенный санузел (типа *bool*). От класса наследуются классы *CountryHouse()* и *Apartment()*. Параметры должны соответствовать требованиям: square не может быть отрицательным, combined_bathroom типа *bool*, иначе вызывается исключение *ValueError* с тестом “Invalid value”.

Класс *CountryHouse()* наследуется от класса *HouseScheme()*. Наследуемые поля: *number_of_rooms*, square, *combined_bathroom*. Поля объекта: *land_are* – площадь участка, number_of_floors – количество этажей. Параметры должны соответствовать требованиям: square не может быть отрицательным, *combined_bathroom* типа *bool*, иначе вызывается исключение *ValueError* с тестом “Invalid value”.

В **классе *CountryHouse()*** переопределяется метод *__str__ (self)*, который возвращает строку: «Количество жилых комнат <количество жилых комнат>,

Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Количество этажей <количество этажей>, Площадь участка <площадь участка>.» Также переопределяется метод *__eq__(self, object)*, который возвращает *True* при равенстве объектов двух классов, в противном случае возвращает *False*.

Класс *Apartment()* наследуется от класса *HouseScheme()*. Наследуемые поля: *number_of_rooms*, *square*, *combined_bathroom*. Поля объекта: *floor* – этаж квартиры, *direction* – куда выходят окна. Параметры должны соответствовать требованиям: floor - число от 1 до 15, значением direction может быть одна из строк: N, S, W, E, иначе вызывается исключение *ValueError* с тестом “Invalid value”.

В **классе *Apartment()*** переопределяется метод *__str__(self)*, который возвращает строку: «Apartment: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Этаж <этаж>, Окна выходят на <куда выходят окна>.»

Класс *CountryHouseList()* наследуется от класса *list*. В конструкторе *__init__(self, name)* инициализируется поле объекта name (строка). В классе переопределен метод *append(self, p_object)*, который добавляет в список *p_object*, если его тип класса *CountryHouse()*, иначе вызывается исключение *TypeError* с тестом “ Invalid type <тип_объекта p_object>”. Также переопределяется метод *total_square(self)*, считающий общую жилую площадь объектов.

Класс *ApartmentList()* наследуется от класса *list*. В конструкторе *__init__(self, name)* инициализируется поле объекта name (строка). В классе переопределен метод *extend(self, iterable)*, который добавляет в список *iterable*, если его тип класса Apartment(), иначе не добавляется. Также переопределяется метод *floor_view(self, floors, directions)*, который печатает на экран объекты, подходящих параметров (этаж попадает в диапазон *floors*, окна выходят на сторону *directions*).

Разработанный программный код см. в приложении А.

Тестирование.

Для тестирования в программу был добавлен код:

```
country_house1 = CountryHouse(6, 100, True, 3, 60)
country_house2 = CountryHouse(3, 45, False, 1, 60)
apartment1 = Apartment(4, 45, True, 4, 'N')
apartment2 = Apartment(4, 45, False, 3, 'N')
print(country_house1)
print(apartment1)
country_list = CountryHouseList("country list")
country_list.append(country_house1)
country_list.append(country_house2)
print(country_list)
print(country_list.total_square)
apartment_list = ApartmentList("apartment list")
apartment_list.extend([apartment1, apartment2])
print(apartment_list)
print(apartment_list.floor_view([3, 5], 'N'))
```

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.		Country House: Количество жилых комнат 6, Жилая площадь 100, Совмещенный санузел True, Количество этажей 3, Площадь участка 60. Apartment: Количество жилых комнат 4, Жилая площадь 45, Совмещенный санузел True, Этаж 4, Окна выходят на N. [<__main__.CountryHouse object at 0x000001FF5CBA3340>, <__main__.CountryHouse object at 0x000001FF5CBA3370>] <bound method CountryHouseList.to-tal_square of [<__main__.CountryHouse object at 0x000001FF5CBA3340>,	Программа работает корректно

		<__main__.CountryHouse object at 0x000001FF5CBA3370>]> [<__main__.Apartment object at 0x000001FF5CBA3400>, <__main__.Apartment object at 0x000001FF5CBA3460>] N: 4 N: 3	

Выводы.

Были изучены принципы объектно-ориентированного программирования. Разработана программа, описывающая классы и их методы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

```
class HouseScheme():
    def __init__(self, number_of_rooms, square, combined_bathroom):
        if square >= 0 and type(combined_bathroom) == bool:
            self.combined_bathroom = combined_bathroom
            self.square = square
            self.number_of_rooms = number_of_rooms
        else:
            raise ValueError('Invalid value')

class CountryHouse(HouseScheme):
    def __init__(self, number_of_rooms, square, combined_bathroom,
number_of_floors, land_area):
        super().__init__(number_of_rooms, square, combined_bathroom)
        if number_of_floors >= 0 and land_area >= 0:
            self.land_area = land_area
            self.number_of_floors = number_of_floors
        else:
            raise ValueError('Invalid value')


    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая площ
адь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.'.format(self.number_of_rooms, self.square, self.combined_bathroom,
self.number_of_floors, self.land_area)

    def __eq__(self, object):
        if self.square == object.square and self.land_area ==
object.land_area and abs(self.number_of_floors - object.number_of_floors)
<= 1:
            return True
        return False

class Apartment(HouseScheme):
    def __init__(self, number_of_rooms, square, combined_bathroom,
floor, direction):
        super().__init__(number_of_rooms, square, combined_bathroom)
        if square >= 0 and type(combined_bathroom) == bool and
(floor >= 1 and floor <= 15) and (direction == 'N' or direction == 'S' or
direction == 'W' or direction == 'E'):
            self.floor = floor
            self.direction = direction
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь
{}, Совмещенный санузел {}, Этаж {}, Окна выходят на
{}.'.format(self.number_of_rooms, self.square, self.combined_bathroom,
self.floor, self.direction)
```

```
    class CountryHouseList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def append(self, p_object):
            if type(p_object) == CountryHouse:
                super().append(p_object)
            else:
                raise TypeError('Invalid type {}'.format(type(p_object)))

        def total_square(self):
            total_area = 0
            for i in range(len(self)):
                total_area += self[i].square
            return total_area

    class ApartmentList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

        def extend(self, iterable):
            super().extend(filter(lambda x: type(x) == Apartment,
iterable))

        def floor_view(self, floors, directions):
            for i in filter(lambda x: floors[0]<= x.floor <= floors[1]
and x.direction in directions, self):
                print ('{}: {}'.format(i.direction, i.floor))
```