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

Студент гр. 1382 _____ Шушков Е. В.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург

2021

Цель работы.

Изучить основы объектно-ориентированной парадигмы программирования. Реализовать систему классов для градостроительной компании на языке Python с использованием основных принципов ООП.

Задание.

Базовый класс -- схема дома 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:

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:

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:

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().

'''

Выполнение работы.

1. HouseScheme - класс-родитель, дочерние классы деревенского дома и апартаментов будут брать из него атрибуты rooms, area, bathroom.

   CountryHouse - класс-наследник HouseScheme, деревенский дом, имеет дополнительные атрибуты floors, garden_area.

   Apartment - класс-наследник HouseScheme, квартира городская, имеет дополнительные атрибуты floor, window.

   CountryHouseList - класс-наследник от list, деревня, т. е. список деревенских домов.

   ApartmentList - класс-наследник от list, жилой комплекс, т. е. список городский квартир.

2. В классах наследников от HouseScheme (CountryHouse и Apartment) были переопределены такие методы, как:

- __str__(), который выводит данные о деревенском доме или квартиры.
- __eq__() в CountryHouse, который сравнивает объекты этого класса между собой по площади вокруг дома и количеству этажей.

От класса list в CountryHouseList были переопределены:

- append(p_object), который выполняет стандартную функцию добавления в конец списка объекта, но теперь при условии, что он является объектом класса CountryHouse. Если объект не этого класса, то выводит ошибку TypeError("Invalid type {}".format(type(p_object))).

От класса list в ApartmentList были переопределены:

- extend(iterable), который также обновляет список, добавляя элементы в конец, но эти элементы должны принадлежать классу Apartment. Если объект не этого класса, то выводит ошибку ValueError("Invalid value").

3. Метод __str__() будет вызван в случаях, когда необходимо будет преобразовать объект с типу str. Например, когда используем функции print().
4. Непереопределенные методы класса list для CountryHouseList и ApartmentList будут работать, потому что все функции класса-родителя работают в классах-наследниках. Например, **list.index**(x) вернёт индекс первого вхождения элемента x в списке, а **list.count**(x) вернёт количество элементов x.

Тестирование.

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарий
1.	`Apt = Apartment(5, 20, True, 16, “N”)`	`builtins.ValueError: Invalid value`	Верный результат

2.	`A=Apartment(5,300,True,4,"W")`
	`B=Apartment(7,300,False,7,"S")`
	`arr = ApartmentList('Hello')`
	`arr.extend([A,B])`
	`print(A)`
	`arr.floor_view([1,6],['W',"N"])`
		`Apartment: Количество жилых комнат 5, Жилая площадь 300, Совмещенный санузел True, Этаж 4, Окна выходят на W.`
		`W: 4`
			Верный результат

Выводы.

Были изучены основы парадигмы программирования, с использованием языка программирования на Python. С помощью ООП была реализована программа, позволяющая создать примитивную базу данных городской или деревенской инфоструктуры.

Приложение А

Исходный код программы

Файл main.py:

```
class HouseScheme:
   def __init__(self, rooms, area, bathroom):
       if area < 0 or not isinstance(bathroom, bool):
           raise ValueError("Invalid value")
       self.rooms = rooms
       self.area = area
       self.bathroom = bathroom


class CountryHouse(HouseScheme):
   def __init__(self, rooms, area, bathroom, floors, garden_area):
       super().__init__(rooms, area, bathroom)
       self.floors = floors
       self.garden_area = garden_area

   def __str__(self):
           return 'Country House: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Количество этажей {}, Площадь участка {}.'.format(
            self.rooms, self.area, self.bathroom, self.floors, self.garden_area)

   def __eq__(self, other):
         if (self.area == other.area and self.garden_area == other.garden_area and
abs(self.floors - other.floors) <= 1):
            return True


class Apartment(HouseScheme):
   def __init__(self, rooms, area, bathroom, floor, window):
       super().__init__(rooms, area, bathroom)
       if not floor in range(1, 16) or not window in "NSWE":
           raise ValueError("Invalid value")
       self.floor = floor
       self.window = window

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

```
            self.rooms, self.area, self.bathroom, self.floor, self.window)


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

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

    def total_square(self):
        return sum(p_object.area for p_object in self)


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

    def extend(self, iterable):
        super().extend(list(filter(lambda i: isinstance(i, Apartment), iterable)))

    def floor_view(self, floors, directions):
        aparts = list(
             filter(lambda apart: apart.floor in range(floors[0], floors[1] + 1) and
apart.window in directions, self))
        for i in aparts:
            print("{}: {}".format(i.window, i.floor))
```