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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Введение в ИТ»
Тема: Парадигмы программирования

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2019

1 Цель работы.

Целью данной работы – научиться работать с **Python** в объектно-ориентированной парадигме программирования.

2 Задание.

Необходимо релизовать систему классов для градостроительной компании.

1. 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**'

2. Дом деревенский **CountryHouse**

   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**'

  Переопределите методы:

- 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**.

3. Квартира городская **Apartment**

   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**'

  Переопределите методы:

- 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:

1. Деревня: реализуйте класс **CountryHouseList**

   Конструктор:

- Вызвать конструктор базового класса
- 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

2. Жилой комплекс: реализуйте класс **ApartmentList**

   Конструктор:

- Вызвать конструктор базового класса
- 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>

  …

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

3 Выполнение работы.

Иерархия классов:

1. **Object**
   - **1.1. HouseScheme**
     - **1.1.1. CountryHouse**
     - **1.1.2. Apartment**
   - **1.2. List**

1.2.1. CountryHouseList

1.2.2. ApartmentList

Переопределенные методы:

init, str, eq, append, extend – переопределены согласно условию

Метод **__str__** вызывается при передаче объектов классов в функцию **print()** или в другие функции, вызывающие его.

Непереопределенные методы класса **list** работают для объектов классов **CountryHouseList** и **ApartmentList.** Это происходит потому, что данные классы наследуются от класса **list**, а следовательно, объекты данных классов будут являтся в то же время объектами класса **list.** При этом нельзя гарантировать, что после применения этих непереопределенных методов, собственные методы классов **CountryHouseList** и **ApartmentList** будут работать корренктно.

Разработанный программный код см. в приложении А.

4 Выводы.

Были изучены основные понятия, и принципы ООП парадигмы написания программ.

Была разработана система классов для градостроительной компании.

1 ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
# -*- coding: utf-8 -*-


class HouseScheme:
    def __init__(self, amount_of_rooms, house_area, combined_bathroom):
        if (amount_of_rooms < 0) or (house_area < 0) or (not isinstance(combined_bathroom, bool)):
            raise ValueError('Invalid value')
        self.amount_of_rooms = amount_of_rooms
        self.house_area = house_area
        self.combined_bathroom = combined_bathroom


class CountryHouse(HouseScheme):
    def __init__(self, amount_of_rooms, house_area, combined_bathroom, amount_of_floors, area):
        super().__init__(amount_of_rooms, house_area, combined_bathroom)
        if (amount_of_floors < 0) or (area < 0):
            raise ValueError('Invalid value')
        self.amount_of_floors = amount_of_floors
        self.area = area

    def __str__(self):
            return 'Country House: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Количество ' \
                   'этажей {}, Площадь участка {}.'.format(self.amount_of_rooms, self.house_area, self.combined_bathroom,
                                                           self.amount_of_floors, self.area)

    def __eq__(self, o):
        """
        :type o: CountryHouse
```

```
        """
        return (self.house_area == o.house_area) and (self.area == o.area) and \
            (abs(self.amount_of_floors - o.amount_of_floors) <= 1)


class Apartment(HouseScheme):
    def __init__(self, amount_of_rooms, house_area, combined_bathroom, floor, windows_direction):
        super().__init__(amount_of_rooms, house_area, combined_bathroom)
        if (floor < 1) or (floor > 15) or (windows_direction not in 'NSWE'):
            raise ValueError('Invalid value')
        self.floor = floor
        self.windows_direction = windows_direction

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Этаж {}, ' \
               'Окна выходят на {}.'.format(self.amount_of_rooms, self.house_area, self.combined_bathroom,
                                            self.floor, self.windows_direction)


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

    def append(self, p_object):
        if not isinstance(p_object, CountryHouse):
            raise TypeError('Invalid type {}'.format(type(p_object)))
        super().append(p_object)

    def total_square(self):
        return sum(map(lambda x: x.house_area, self))
```

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

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

    def floor_view(self, floors, directions):
        for x in filter(lambda flat: (flat.floor in range(floors[0], floors[1] + 1)) and (
                                    flat.windows_direction in directions), self):
            print("{}: {}".format(x.windows_direction, x.floor))
```