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. 1384 _____ Усачева Д.В.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург

2021

Цель работы.

Разработать систему классов для потенциального использования в градостроительной компании.

Задание.

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().
        '''
```

В отчете укажите:

1. Иерархию описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будет вызван метод __str__().
4. Будут ли работать непереопределенные методы класса list для CountryHouseList и ApartmentList? Объясните почему и приведите примеры.

Выполнение работы.

Иерархия классов:

HouseScheme :

- CountryHouse
- Apartment

List :

- CountryHouseList
- ApartmentList

Согласно условию, создан базовый класс для деревенского дома и квартиры HouseScheme. В конструкторе этого класса принимается натуральное число living_rooms – количество жилых комнат, целое неотрицательное число living_space – жилая площадь помещения, а также логическая переменная combined_bathroom. Перед сохранением значения в соответствующие поля объекта, проверяется их корректность.

После был реализован класс для деревенского дома CountryHouse. Наследование позволяет избежать повторения кода (сохранения количества жилых комнат, жилплощади и совмещенности санузла). Конструктор проверяет значения параметров, характерных конкретно для этого класса и

аналогично выбрасывает исключение, если какой-то из этих параметров некорректен. В ином случае он сохраняет их как поля объекта. У этого класса был переопределен метод __str__, преобразующий объект в строку согласно заданию. Также был переопределен метод __eq__, позволяющий сравнивать два объекта этого класса согласно заданным условиям.

Далее был создан класс для городской квартиры Apartment. Наследование позволяет избежать повторения кода (сохранения количества жилых комнат, жилплощади и совмещенности санузла). В конструкторе класса Apartment сначала вызывается конструктор класса HouseScheme, куда передаются эти три параметра. Конструктор класса Apartment проверяет значения параметров, характерных конкретно для этого класса, выбрасывая исключение, если какой-то из этих параметров некорректен. В ином случае он сохраняет их как поля объекта. У этого класса метод __str__ переопределен согласно условию, он преобразует объект в строку как это указано в задании.

Далее был создан класс для работы с деревенскими домами CountryHouseList. В конструкторе этого класса принимается строка name и она сохраняется как поле. В этом классе был переопределен метод append. В нем перед добавлением переменной (вызов append класса-родителя) выполняется проверка, является ли эта переменная объектом класса CountryHouse. Также был создан метод total_square, подсчитывающий общую жилплощадь содержащихся в этом списке деревенских домов.

Далее был создан класс для жилого комплекса ApartmentList. В конструкторе этого класса принимается строка name и она сохраняется как поле. В этом классе был перезаписан метод extend. В нем перед добавлением элемента (вызов append класса-родителя) выполняется проверка, является ли этот элемент объектом класса Apartment. Также был создан метод floor_view, выводящий на экран все квартиры, находящиеся между двумя этажами, переданными первым аргументом, и окна которых выходят в одном из направлений, переданных вторым аргументом.

Разработанный программный код см. в приложении 1.

Выводы

Были изучены различные парадигмы программирования, а также механизм наследования. Разработана система классов.

Приложение 1

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, living_rooms, living_space, combined_bathroom):
        if isinstance(living_rooms, (int, float)) and isinstance(living_space,
(int, float)) \
                and living_space >= 0 and isinstance(combined_bathroom, bool):
            self.living_rooms = living_rooms
            self.living_space = living_space
            self.combined_bathroom = combined_bathroom
        else:
            raise ValueError('Invalid value')


class CountryHouse(HouseScheme):
    def __init__(self, living_rooms, living_space, combined_bathroom,
floors_num, land_area):
        super().__init__(living_rooms, living_space, combined_bathroom)
        if isinstance(floors_num, (int, float)) and isinstance(land_area, (int,
float)):
            self.floors_num = floors_num
            self.land_area = land_area
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        tx = [self.living_rooms, self.living_space, self.combined_bathroom,
self.floors_num, self.land_area]
        return 'Country House: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Количество этажей {}, Площадь участка {}.'.format(
            *tx)

    def __eq__(self, other):
        if not isinstance(other, CountryHouse):
            raise ValueError('Invalid value')

        return self.living_space == other.living_space and self.land_area ==
other.land_area and \
               abs(self.floors_num - other.floors_num) <= 1


class Apartment(HouseScheme):
    def __init__(self, living_rooms, living_space, combined_bathroom, floor,
windows_direction):
        super().__init__(living_rooms, living_space, combined_bathroom)
        if isinstance(floor, (int, float)) and floor >= 1 and floor <= 15 and
['N', 'S', 'W', 'E'].__contains__(
                windows_direction):
            self.floor = floor
            self.windows_direction = windows_direction
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        tx = [self.living_rooms, self.living_space, self.combined_bathroom,
self.floor, self.windows_direction]
```

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


class CountryHouseList(list):
    def __init__(self, name):
        super().__init__()
        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):
        all_living_space = 0
        for i in self:
            all_living_space += i.living_space
        return all_living_space


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

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

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