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

ОТЧЕТ

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

Студентка гр. 1384 _____ Логинова А. Ю.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2021

Цель работы.

Научиться работать с классами в Python, ознакомиться с объектно-ориентированной парадигмой программированием.

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 повторяться. 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.

В программе были реализованы классы для градостроительной компании: HouseScheme, CountryHouse, Apartment, CountryHouseList, ApartmentList.

Класс *HouseScheme* принимает на вход данные о: количестве комнат (*amount_of_rooms*), величине жилой площади (*house_area*), наличии совместного санузла (*combined_bathroom*). Если данные были введены корректно, а именно неотрицательное значение house_area и значение типа *bool combined_bathroom,* конструктор создаст внутренние переменные для экземпляра класса. Если данные были введены некорректно, программа выдаст ошибку и прекратит работу.

Класс *CountryHouse* принимает на вход данные о: количестве комнат (*amount_of_rooms*), величине жилой площади (*house_area*), наличии совместного санузла (*combined_bathroom*), количестве этажей (*amount_of_floor*), величине всей площади (*area*). Класс наследуется от *HouseScheme*, в конструкторе которого первые три аргумента проверяются на корректность. Класс обладает методами *__str__()* и *__eq__()*. Метод *__str__()* возвращает данные об экземпляре в виде строки. Метод *__eq__()* сравнивает два экземпляра, возвращает значение True, если у экземпляров одинаковая величина жилой и общей площади, а разница в количестве этажей не превышает единицы и возвращает False в противном случае.

Класс *Apartment* принимает данные на вход о: количестве комнат (*amount_of_rooms*), величине жилой площади (*house_area*), наличии

совместного санузла (*combined_bathroom*), этаже (*floor*), стороне, на которую выходит окно (*window_side*). Если данные были введены корректно, а именно значение *floor* в диапазоне от 1 до 15 и значение *window_side* одна из литер: N, W, E, S, конструктор создаст внутренние переменные для экземпляра класса. Так как класс наследуется от HouseScheme, первые три величины проходят проверку внутри его конструктора. Если данные были введены некорректно, программа выдаст ошибку и прекратит работу. Класс обладает методом *__str__()*, который возвращает данные об экземпляре класса в виде строки.

Класс CountryHouseList принимает на вход название для будущего списка из объектов. Класс наследуется от List. В классе реализовано переопределение метода *append()*, который добавляет объект в список только если тип объекта CountryHouse. Также был реализован метод *total_square()*, который суммирует площадь всех объектов в списке.

Класс ApartmentList принимает на вход название для будущего списка из объектов. Класс наследуется от List. В классе реализовано переопределение метода *extend()*, который добавляет объект в список только если тип объекта Apartment. Также был реализован метод *floor_view()*, который выводит информацию об этаже и стороне, на которое выходит окно, в зависимости от переданных аргументов – диапазона этажей и сторон.

1. Иерархия описанных классов:
    - List
        - ApartmentList
        - CountryHouseList
    - HouseScheme
        - Apartment
        - CountryHouse
2. Переопределенные методы: *list.append(), list.extend(), object.__init__(), object.__str__(), object.__eq__()*

3. При приведении экземпляра класса Apartment к классу str: *str(Apartment), Apartment.__str__()*
4. Непереопределенные методы будут работать, т.к. они будут вызываться уже не в, например, ApartmentList, а в List, потому как класс от него наследуется, а следовательно имеет доступ к его методам. Например, метод clear(), который не был переопределен нигде, очистит списки.

   Разработанный программный код приводится в приложении А.

Выводы.

Были изучены возможности объектно-ориентированного программирования на Python, написана программа, реализующая классы для градостроительной компании.

ПРИЛОЖЕНИЕ А.

Исходный код программы.

Название файла: main.py

```
    def __init__(self, amount_of_rooms, house_area, combined_bathroom):
        if not (house_area >= 0 and 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_floor, area):
        super().__init__(amount_of_rooms, house_area, combined_bathroom)
        self.amount_of_floor = amount_of_floor
        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_floor, self.area)

    def __eq__(self, other):
        if self.house_area == other.house_area and self.area ==
other.area \
                and abs(self.amount_of_floor - other.amount_of_floor) <=
1:
            return True
        return False


class Apartment(HouseScheme):
    def __init__(self, amount_of_rooms, house_area, combined_bathroom,
floor, window_side):
        super().__init__(amount_of_rooms, house_area, combined_bathroom)
        if not (0 < floor < 16 and window_side in ('N', 'S', 'W', 'E')):
            raise ValueError('Invalid value')
        self.floor = floor
        self.window_side = window_side

    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.window_side)


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

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

    def total_square(self):
        overall_area = 0
        for house in self:
            overall_area += house.house_area
        return overall_area


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

    def extend(self, iterable):
        for item in iterable:
            if isinstance(item, Apartment):
                super().append(item)

    def floor_view(self, floors, directions):
        for apartment in list(filter(lambda item: max(floors) >=
int(item.floor) >= min(floors) and
                                                    item.window_side in
directions, self)):
            print('{}: {}'.format(apartment.window_side,
apartment.floor))
```