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. 9381 _____ Колованов Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2019

Цель работы.

Изучение парадигм программирования, а также использование парадигмы ООП для написания программ.

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(). '"

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.

На схеме 1 представлена иерархия описанных классов:

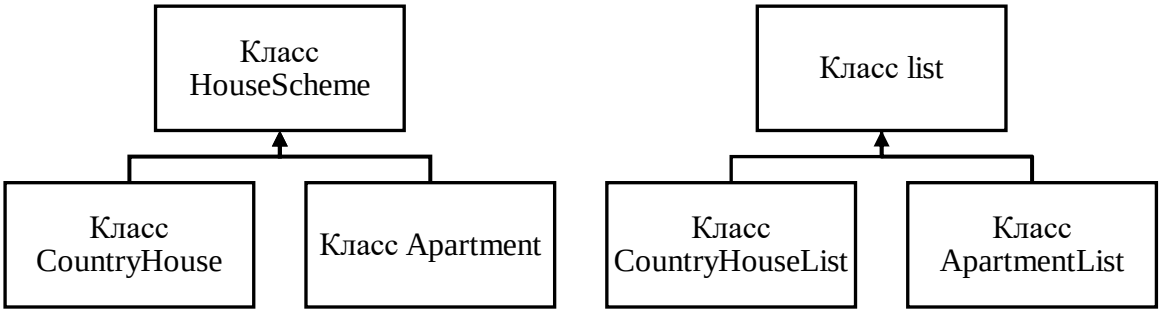

Схема 1. Система классов для градостроительной компании

Класс *HouseScheme* имеет следующие поля:

- *room* – количество комнат.
- *area* – площадь.
- *bathroomAvailability* – наличие совмещенного санузла.

В конструкторе происходит проверка данных на корректность. Был переопределен метод __init__().

Класс *CountryHouse* имеет следующие поля:

- Поля, наследуемые из класса *HouseScheme*.
- *floors* – количество этажей.
- *landArea* – площадь участка.

В конструкторе происходит проверка данных на корректность. Были переопределены методы *__str__(), __eq__()* и *__init__()*..

Класс *Apartment* имеет следующие поля:

- Поля, наследуемые из класса *HouseScheme*.
- *floor* – этаж.

- *side* – сторона выхода окон.

В конструкторе происходит проверка данных на корректность. Были переопределены методы *__str__()* и *__init__()*.

В классе *CountryHouseList* переопределен метод *append()*, а также написан метод *total_square()*.

В классе *ApartmenList* переопределен метод *extend()*, а также написан метод *floor_view()*.

Метод *__str__()* вызывается при преобразовании к строке при помощи функции *str()*, при выводе на экран при помощи функции *print()*, а также при передаче объекта как аргумента функции *format()*. Если *__str__()* не определен, но *__repr__()* определен, то будет вызван метод *__repr__()*.

Не переопределенные методы класса *list* могут быть вызваны объектами производных классов. В данном случае, если мы попробуем вызвать у объекта класса *CountryHouseList* метод *count()*, то будет вызван метод *count()* класса *list*, так как мы унаследовали его. К примеру:

```
object = CountryHouseList()
print(object.count()) # Выведет 0
```

Разработанный программный код см. в приложении А.

Тестирование.

Тестирование производилось на образовательной платформе Stepik.

Выводы.

Были изучены основные парадигмы программирования. На языке Python, руководствуясь парадигмой ООП, была разработана программа,

предоставляющая систему классов для градостроительной компании. Для написания программы использовались условные операторы, операторы цикла, классы, функция *filter()*, а также обработчик исключений *try-except*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, rooms, area, bathroomAvailability):
        if (area < 0) or ((bathroomAvailability is not True) and
(bathroomAvailability is not False)):
            raise ValueError("Invalid value")

        self.rooms = rooms
        self.area = area
        self.bathroomAvailability = bathroomAvailability


class CountryHouse(HouseScheme):
    def __init__(self, rooms, area, bathroomAvailability, floors,
landArea):
        HouseScheme.__init__(self, rooms, area, bathroomAvailability)
        self.floors = floors
        self.landArea = landArea

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

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


class Apartment(HouseScheme):
    def __init__(self, rooms, area, bathroomAvailability, floor,
side):
        HouseScheme.__init__(self, rooms, area, bathroomAvailability)
        self.floor = floor
        self.side = side

        if self.floor < 1 or self.floor > 15:
            raise ValueError("Invalid value")

        if self.side not in ["N", "S", "W", "E"]:
            raise ValueError("Invalid value")

    def __str__(self):
        return "Apartment: Количество жилых комнат {}, Жилая площадь
{}, Совмещенный санузел {}, Этаж {}, Окна выходят на
{}.".format(self.rooms, self.area, self.bathroomAvailability,
self.floor, self.side)


class CountryHouseList(list):
```

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

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

    def total_square(self):
        return sum([x.area for x in self])


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

    def extend(self, iterable):
        filteredIterable = filter(lambda x: isinstance(x, Apartment),
iterable)
        list.extend(self, filteredIterable)

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