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. 0304 _____ Докучаев Р.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2020

Цель работы.

Изучить создание собственных классов и работу с ними. Изучить наследование классов и их иерархию.

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().'"

Основные теоретические положения.

Были использованы *иерархия классов и их наследование*. При определении метода, присутствовавшего в родительском классе, в дочернем классе происходит *переопределение*. Для вызова конструктора была использована конструкция *__init__()*. Для вызова метода родителя внутри дочернего класса была использована функция *super()*. Метод *__str__()* будет вызван при попытке вывести объект (или экземпляр) данного класса. В случае, если класс наследуется от другого класса, но при этом некоторые методы родительского класса не были в нём переопределены, то при попытке вызвать метод для данного экземпляра класса будет вызван метод родительского класса. Для вывода исключений была использована команда *raise*. Также были использованы *базовые конструкции языка Python*.

Выполнение работы

1. Был создан класс *HouseScheme*, который имеет поля *rooms* (количество комнат), *area* (жилая площадь), *bathroom* (наличие раздельного санузла). В конструкторе *__init__* полям были присвоены соответствующие значения *self.rooms, self.area* и *self.bathroom*. Далее данные поля были проверены на соответствие условиям задания при помощи *isinstance()*. В случае, если условия, описанные в программе, не выполняются, Python выдаёт исключение *ValueError*.
2. Был создан *дочерний для HouseScheme класс CountryHouse*. Были переопределены методы *__str__ и __eq__*. В конструкторе классу-родителю была передана часть данных родительского класса, также были инициализированы ещё два поля *plot_area* (площадь участка) и *floors* (количество этажей). Эти поля были проверены на соответствие условиям задачи с помощью *__eq__*. Если обнаружено несоответствие, генерируется исключение *ValueError('Invalid value')*.
3. Был создан класс *Apartment, наследник класса HouseScheme*. Для данного класса был переопределен метод *__str__*. В конструкторе из родительского *HouseScheme* была передана часть аргументов, также были инициализированы два новых поля *big_floors* (этаж, между 1 и 15) и *windows* (сторона, на которую

выходят окна. Эти поля были проверены на соответствие условиям с помощью *__eq__*. Если найдена ошибка, генерируется исключение *ValueError('Invalid value')*.

4. Был создан класс *CountryHouseList*, являющийся дочерним для класса *list*. Был переопределён метод *append* таким образом, чтобы в список добавлялись только элементы, соответствующие классу *CountryHouse*. Если объект относится к другому классу, будет сгенерировано исключение *TypeError*. Был объявлен метод *total_square*, возвращающий суммарную жилую площадь.
5. Был создан класс *ApartmentList*, который является дочерним классом от *list*. Был переопределён метод *extend* так, чтобы в список добавлялись только элементы класса *Apartment*. Был объявлен метод *floor_view*, который при помощи функции *filter()* и *лямбда-выражения* производит отбор элементов из списка согласно условию.

Разработанный программный код см. в приложении А.

Выводы.

Были изучены парадигмы программирования.

Была разработана программа, которая позволяет создавать объекты определённых классов и работать с ними. Для проверки принадлежности переменной конкретному классу были использованы специализированные функции *filter* (для фильтрации) и *isinstance* (для проверки принадлежности к определенному классу), а также *lambda-выражение*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: laba3.py

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


class CountryHouse(HouseScheme):
    def __init__(self, rooms, area, bathroom, floors,
plot_area):
        super().__init__(rooms, area, bathroom)
        self.floors = floors
        self.plot_area = plot_area
        if not isinstance(self.floors, int) or not
isinstance(self.plot_area, int):
            raise ValueError('Invalid value')


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


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


class Apartment(HouseScheme):
    def __init__(self, rooms, area, bathroom,
big_floors, windows):
        super().__init__(rooms, area, bathroom)
        if isinstance(big_floors, int) and windows in
['N', 'W', 'S', 'E'] and big_floors in range(1, 16):
            self.big_floors = big_floors
            self.windows = windows
        else:
```

```
            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.bathroom,
self.big_floors, self.windows)


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 {}'.for-
mat(type(p_object)))

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


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

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

    def floor_view(self, big_floors, directions):
        filter_function = lambda obj: obj.big_floors
in range(big_floors[0], big_floors[1] + 1) and
obj.windows in directions
        result = list(filter(filter_function, self))
        for element in result:
            print('{}: {}'.format(element.windows,
element.big_floors))
```