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

Студентка гр. 0382 _____ Деткова А.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Изучить парадигмы программирования (императивная, декларативная и их подвиды). Уточнить работу с ООП. Научиться создавать классы самостоятельно. Обучиться работе с исключениями.

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()*.

Основные теоретические сведения.

Парадигма программирования — совокупность идей и понятий, определяющих стиль написания программ (подход к программированию).

Виды: императивная и декларативная.

Объектно-ориентированное программирование:

- Программа — совокупность объектов.
- Объект — экземпляр класса.
- Класс — описание некоторого типа данных.
- Классы используют иерархию наследования.

Метод класса — функция, которая принадлежит классу.

Конструктор — метод, который вызывается при создании экземпляра класса (объекта класса).

Поле — некоторая переменная, которая лежит в области видимости объекта (т. е. принадлежит объекту).

Принципы ООП:

- Инкапсуляция
- Наследование
- Полиморфизм

Ошибки и исключения в Python — особый класс объектов в Python, возникают из-за неверного синтаксиса, ошибки в коде, а также из-за генерации ошибки вручную.

Выполнение работы.

1. Иерархия классов:

   CountryHouse от *HouseScheme*, *Apartment* от *HouseScheme*, *CountryHouseList* от *list*, *Apartment* от *list*.

2. В классе *CountryHouse* переопределены:

__str__() - возвращает всю информацию об объекте класса (количество жилых комнат, жилая площадь, наличие совмещенного санузла, количество этажей, площадь участка).

__eq__() - позволяется сравнивать объекты класса друг с другом, возвращает *True*, если жилая площадь и площадь участка равны, а количество этажей отличается не больше, чем на 1.

В классе *Apartment* переопределены:

__str__() - возвращает всю информацию об объекте класса (количество жилых комнат, жилая площадь, наличие совмещенного санузла, этаж, сторона света, куда выходят окна).

В классе *CountryHouseList* переопределены:

append(p_object) - работает как *append* базового класса, но в конец списка добавляются только объекты класса *CountryHouse*.

total_square() - находит и возвращает общую жилую площадь домов.

В классе *Apartment* переопределены:

extend(iterable) - работает как *extend* базового класса, но в конец списка добавляются только объекты из списка класса *Apartment.*

floor_view(floors, directions) — возвращает список этаж: направление, где для каждой квартиры из списка проверяется принадлежит ли ее этаж этажу из диапазона *floors* и направление из *directions.*

3. Метод *__str__()* вызывается, когда нужно вывести какое-то сообщение на экран, сначала вызывается метод *str,* который преобразует сообщение в строку и после оно выводится на экран функцией *print*. Также этот метод вызывается в функции *str,* в методе *format.*

4. Да, будут работать, т. к. *CountryHouseList* и *ApartmentList* являются наследниками класса *list,* а значит, могут использовать все методы родительского класса. Например, будет спокойно работать функция сложения списков, которая объединит 2 списка в 1, подставив 1 в конец другого.

Разработанный программный код см. в приложении А.

Выводы.

Изучили более подробно объектно-ориентированную парадигму. Научились работать с классами, их методами и полями. Изучили особый класс — исключения. Познакомились с другими парадигмами программирования.

Была разработана программа, которая описывается структуру градостроительной компании, ее классы, а также атрибуты, присущие каждому классу.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

```
class HouseScheme:
    def __init__(self, Rooms, Square, Bath):
        if (Square > 0) and (Rooms > 0) and (type(Bath) == bool):
            self.Rooms = Rooms
            self.Square = Square
            self.Bath = Bath
        else:
            raise ValueError('Invalid value')

class CountryHouse(HouseScheme):
    def __init__(self, Rooms, Square, Bath, Floors, Plottage):
        super().__init__(Rooms, Square, Bath)
        if (Floors > 0) and (Plottage > 0):
            self.Floors = Floors
            self.Plottage = Plottage
        else:            raise ValueError('Invalid value')
    def __str__(self):
         return "Country House: Количество жилых комнат {}, Жилая пло-
щадь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.".format(self.Rooms,    self.Square,    self.Bath,    self.Floors,
self.Plottage)
    def __eq__(self, other):
             if (self.Square == other.Square) and (self.Plottage ==
other.Plottage) and (abs(self.Floors - other.Floors) <= 1):
            return True
        else:
            return False

class Apartment(HouseScheme):
    def __init__(self, Rooms, Square, Bath, Floors, Windows):
        super().__init__(Rooms, Square, Bath)
        if (Floors <= 15) and (Floors >= 1) and (Windows in ['N', 'S',
'W', 'E']):
            self.Floors = 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.Square,    self.Bath,    self.Floors,
self.Windows)

class CountryHouseList(list):
    def __init__(self, name):
        super().__init__(self)
        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):
        ls = list(map(lambda x: x.Square, self))
        return sum(ls)

class ApartmentList(list):
    def __init__(self, name):
        super().__init__(self)
        self.name = name
    def extend(self, iterable):
         super().extend(list(filter(lambda x: isinstance(x, Apartment),
iterable)))
    def floor_view(self, floors, directions):
            res = list(filter(lambda x: (x.Floors <= floors[1]) and
(x.Floors >= floors[0]) and (x.Windows in directions), self))
        for i in res:
            print("{}: {}".format(i.Windows, i.Floors))
```