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. 0382 _____ Гудов Н.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Изучение парадигмы ООП на языке Python. Создание системы классов для градостроительной компании на языке программирования Python

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 комнатажилая 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

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: class CountryHouseList: # список деревенских домов -- "деревня", наследуется от класса list

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>'"

Жилой комплекс: class ApartmentList: # список городских квартир -- ЖК, наследуется от класса list Конструктор. Вызвать конструктор базового класса 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. Направления и этажи могут повторятся. Для реализации используйте функцию filter().

Основные теоретические положения.

Объектно-ориентированное программирование (ООП) — это парадигма программирования, где различные компоненты компьютерной программы моделируются на основе реальных объектов. Объект — это что-либо, у чего есть какие-либо характеристики и то, что может выполнить какую-либо функцию.

Выполнение работы.

Созданы следующие элементы

Класс HouseScheme. Принимает основные параметры дома, записывает их значения в поля класса

Поля: .rooms .place .bath- кол-во комнат, площадь внутри, санузел

Класс CountryHouse, наследник HouseScheme. Принимает параметры и сравнивает дома, по полученным измерениям.

Поля: .floors .square-этажи, площадь участка

Класс Apartment, наследник HouseScheme

Поля: .side .floor- сторона окон и этаж

Класс CountryHouseList, наследник List

Класс ApartmentList, наследник List, имеет метод, печатающий нужные квартиры.

Также переопределялись некоторые методы базовых классов

Выводы.

Были изучены парадигмы программирования в Python. Полученная информация была успешно применена на практике. Разработана программа, содержащая систему классов градостроительной компании.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла menu3.py

```
class HouseScheme:
    def __init__(self, rooms, place, bath):
        if type(bath) == bool and place >= 0:
            self.rooms = rooms
            self.place = place
            self.bath = bath
        else:
            raise ValueError("Invalid value")


class CountryHouse(HouseScheme):
    def __init__(self, rooms, place, bath, floors, square):
        if place > 0 and type(bath) == bool:
            super().__init__(rooms, place, bath)
            self.floors = floors
            self.square = square
        else:
            raise ValueError("Invalid value")
    def __str__(self):
        return "Country House: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка {}.".format(
            self.rooms, self.place, self.bath, self.floors, self.square)

    def __eq__(self, other):
        if type(other) != CountryHouse: raise ValueError("Invalid value")
        return self.place == other.place and self.square == other.square and abs(self.floors - other.floors) <= 1


class Apartment(HouseScheme):

    def __init__(self, rooms, place, bath, floor, side):
```

```
            super().__init__(rooms, place, bath)
            if 1 <= floor <= 15 and side in ['N', 'S', 'W', 'E']:
                self.floor = floor
                self.side = side
            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.place, self.bath, self.floor, self.side)


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

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

        def total_square(self):
            total_place = 0
            for i in range(len(self)):
                total_place += self[i].place
            return total_place


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

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

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