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 _____ Мукатанов А.В.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Создать систему классов для градостроительной компании, используя парадигму ООП на языке Python.

Задание.

Создать систему классов для градостроительной компании: HouseScheme - базовый класс (схема дома); • Поля класса: • количество жилых комнат • жилая площадь (в квадратных метрах) • совмещенный санузел (значениями могут быть или False, или True) • При несоответствии переданного значения вызвать исключение. CountryHouse — деревенский дом (наследник HouseScheme); Поля класса:

все поля класса HouseScheme

количество этажей

площадь участка При несоответствии переданного значения вызвать исключение.

Переопределяемые методы:

__str__()

__eq__()

Apartment — квартира городская (наследник HouseScheme); Поля класса:

все поля класса HouseScheme

этаж (может быть число от 1 до 15)

куда выходят окна (значением может быть одна из строк: N, S, W, E))

При несоответствии переданного значения вызвать исключение. Определяемые методы:

__str__() CountryHouseList — список деревенских домов (наследник list);

3 Конструктор: Вызвать конструктор базового класса a. Передать в конструктор строку name и присвоить её полю name созданного объекта

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

total_square(): Посчитать общую жилую площадь. 1. ApartmentList — список городских квартир (наследник list);

Методы: Конструктор:

a. Вызвать конструктор базового класса

b. Передать в конструктор строку name и присвоить её полю name

c. созданного объекта

extend(iterable):

Переопределение метода extend() списка. В случае, если элемент b. iterable - объект класса Apartment, этот элемент добавляется в c. список, иначе не добавляется.

floor_view(floors, directions): В качестве параметров метод получает диапазон возможных этажей в виде списка. Метод должен выводить квартиры, этаж которых входит в диапазон и окна которых выходят в одном из направлений.

Основные теоретические положения.

Объектно-ориентированная парадигма базируется на нескольких принципах: наследование, инкапсуляция, полиморфизм. Наследование - специальный механизм, при котором мы можем расширять классы, усложняя их функциональность. В наследовании могут участвовать минимум два класса: суперкласс (или класс-родитель, или базовый класс) - это такой класс, который был расширен. Все расширения, дополнения и усложнения класса-родителя реализованы в классе-наследнике (или производном классе, или классе потомке) - это второй участник механизма наследования. Наследование позволяет повторно использовать функциональность базового класса, при этом не меняя базовый класс, а также расширять ее, добавляя новые атрибуты.

Выполнение работы.

Был создан главный класс HouseScheme, от когорого будут наследоваться классы CountryHouse и Apartment. Далее был определен конструктор с учетом принимаемых аргументов. Аргументы были проверены на соответствие типов и значений и записаны в соответствующие поля объекта класса. При несоответствии типов или значений будет выброшено исключение ValueError с

сообщением «Invalid value». Был создан класс CounrtyHouse , наследующийся от класса HouseScheme. В нем был определен конструктор с использованием конструктора класса - родителя. Соответствующие аргументы записываются в соответствующие поля. Далее был переопределен метод __str__ и __eq__ в соответствии с условием задания. Был создан класс Apartment, наследующийся от класса HouseScheme. Было определено поле класса с возможными направлениями выхода окон. Был определен конструктор с использованием конструктора класса — родителя. Аргументы были проверены критериями из условия и записаны в соответствующие поля. При несоответствии типов или значений будет выброшено исключение ValueError с сообщением «Invalid value». Был переопределен метод __str__ в соответствии с условием. Был создан класс CountryHouseList, наследующийся от стандартного класса list. Был определен конструктор с использованием конструктора класса — родителя. Аргументы были записаны в соответствующие поля. Был переопределен метод append в соответствии с условием и был определен метод total_area, вычисляющий общую жилую площадь. Был создан класс ApartmentList, наследующийся от стандартного класса list. В нем был определен конструктор с использованием конструктора класса — родителя. Был переопределен метод extend в соответствии с условиями и определен метод floor_view, который выводит этажи и направления окон квартир, соответствующим условиям аргументов.

Разработанный код см. в приложении А.

Выводы.

В ходе работы было выполнено задание — построена система классов для градостроительной компании. Реализована система классов, представлена иерархия классов, переопределены все необходимые методы классов, инициализированы поля объектов классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла main.py

```
class HouseScheme:
    def __init__(self, num_of_room, area, comb_b):
        if area < 0 or type(combwc)!=bool:
            raise ValueError("Invalid value")
        self.num_of_room = num_of_room
        self.area = area
        self.comb_b = comb_b


class CountryHouse(HouseScheme):
    def __init__(self, num_of_room, area, comb_b, num_of_floors, land_area):
        super().__init__(num_of_room, area, comb_b)
        self.num_of_floors = num_of_floors
        self.land_area = land_area


    def __str__(self):
        return "Coun-
try House: Количество жилых комнат {}, Жилая площадь {}, Совмещенный сан
узел {}, Количество " \
               "этажей {}, Пло-
щадь участка {}.".format(self.num_of_room, self.area, self.comb_b,self.num_of_flo
ors, self.land_area)

    def __eq__(self, other):
        re-
turn self.area == other.area and self.land_area == other.land_area and abs(self.num_o
f_floors - other.num_of_floors) <= 1

class Apartment(HouseScheme):
    window_directions = ['N', 'S', 'W', 'E']

    def __init__(self, num_of_room, area, comb_b, floor, side):
        super().__init__(num_of_room, area, comb_b)
        if (1 <= floor <= 15) and (side in Apartment.window_directions):
            self.floor = floor
            self.side = side
        else:
            raise ValueError('Invalid value')

    def __str__(self):
```

```
        return 'Apart-
ment: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел
{}, Этаж {}, ' \
               'Окна выходят на {}.'.for-
mat(self.num_of_room, self.area, self.comb_b, self.floor, self.side)


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 {}".format(type(p_object)))

    def total_square(self):
        res = 0
        for house in self:
            res+=house.area
        return res


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

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

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