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. 1382 _____Коренев Д.А.
Преподаватель _____Шевская Н.В.

Санкт-Петербург
2021

Цель работы.

Изучить основы объектно-ориентированной парадигмы программирования. Реализовать систему классов для градостроительной компании на языке Python с использованием основных принципов ООП.

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 , Жилая площадь , Совмещенный санузел , Количество этажей , 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: Вызвать конструктор базового класса, 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

Метод total_square(): Посчитать общую жилую площадь

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. Формат вывода:

<Направление_1>: <этаж_1>

<Направление_2>: <этаж_2>

...

Направления и этажи могут повторятся. Для реализации используйте функцию filter().

Основные теоретические положения.

- Одна из самых популярных реализаций функционального программирования во многих языках программирования – это лямбда выражения. Лямбда-выражения – это специальный элемент синтаксиса для 5 создания анонимных (т.е. не имеющих имени) функций сразу в том месте, где эту функцию необходимо вызвать. Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций. Синтаксис определения следующий: lambda аргумент1, аргумент2,..., аргументN : выражение

- Функция filter(): Синтаксис функции: filter(,) Функция применяется для каждого элемента итерируемого объекта и возвращает объект-итератор, состоящий из тех элементов итерируемого объекта , для которых является истиной.
- Синтаксис создания класса:

class <Название класс>:

<Тело класса>

- Синтаксис конструктора:

def __init__(self, <аргумент_1>, …, <аргкмент_N>):

<Тело конструктора класса>

- 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.

Синтаксис:

class A: # класс-родитель

<Тело класса>

class B(A): # класс-потомок

<Тело класса>

- isinstance(obj_, class_) Функция возвращает True, если obj_ является экземпляром класса class_ или если class_ является суперклассом для класса, объектом которого является obj_.

Выполнение работы.

- Иерархия описанных классов:

Класс-родитель : HouseScheme; Классы-наследники класса HouseScheme: CountryHouse, Apartment

Класс-родитель : list; Классы-наследники класса list: CountryHouseList, ApartmentList.

- В дочерних классах класса HouseScheme (CountryHouse и Apartment) был переопределен метод__str__(), которые выводят параметры дома и квартиры. Также в классе CountryHouse был переопределён метод __eq__(), который сравнивает объекты типа CountryHouse по площади и кол-ву этажей.
- В классе list были переопределены следующие методы: у класса-наследника CountryHouseList – append(p_object), у которого в случае, если p_object - деревенский дом, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>; у класса-наследника ApartmentList - extend(iterable), у которого в случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.
- Метод __str__() будет вызван в случаях, когда будет необходимо преобразование объекта к типу str.
- Непереопределённые методы класса list будут работать в дочерних классах CountryHouseList и ApartmentList, так как все методы класса-родителя работают в классах-наследниках. Например self.pop() – удалит и вернет последний элемент списка, self.reverse() – перевернут список (последний элемент будет первый, предпоследний – вторым и тд.).

Тестирование.

Тестирование представлено в табл. 1.

Код для теста №1 (code_1):

```
obj_1 = CountryHouse(3, 90, True, 1, 300)
obj_2 = CountryHouse(5, 150, False, 2, 400)
arr = CountryHouseList("CH")
arr.append(obj_1)
arr.append(obj_2)
print(arr.total_square(), '\n', obj_1, '\n', obj_2)
```

Код для теста №2 (code_2):

```
obj_1 = Apartment(2, 80, True, 6, 'N')
obj_2 = Apartment(3, 135, False, 12, 'W')
arr = ApartmentList("AH")
AL = [obj_1, obj_2]
arr.extend(AL)
arr.floor_view([1, 10], ['N', 'E', 'W'])
```

Таблица 1 – Тестирование

№ п/п	Входные данные	Выходные данные	Комментарии
1	code_1	240 Country House: Количество жилых комнат 3, Жилая площадь 90, Совмещенный санузел True, Количество этажей 1, Площадь участка 300. Country House: Количество жилых комнат 5, Жилая площадь 150, Совмещенный санузел False, Количество этажей 2, Площадь участка	Программа работает правильно
2	code_2	N: 6	Программа работает правильно

Вывод.

Были изучены основы объектно-ориентированного программирования. Реализована система классов для градостроительной компании с использованием основных принципов ООП, включающая классы HouseScheme, CountryHouse, Apartament, CountryHouseList, ApartamentList.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Korenev_Danil_lb3.py

```
class HouseScheme:

    def __init__(self, rooms, square, bathroom):
        if isinstance(rooms, int) and rooms > 0 and isinstance(square,
int) and square > 0 and isinstance(bathroom,

bool):
            self.rooms = rooms
            self.square = square
            self.bathroom = bathroom
        else:
            raise ValueError("Invalid value")


class CountryHouse(HouseScheme):

    def __init__(self, rooms, square, bathroom, floors, area):
        if isinstance(floors, int) and floors > 0 and isinstance(area,
int) and area > 0:
            super().__init__(rooms, square, bathroom)
            self.floors = floors
            self.area = area
        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.square, self.bathroom, self.floors,
self.area)

    def __eq__(self, other):
        if isinstance(other, CountryHouse):
            if self.square == other.square and self.area == other.area
and abs(self.floors - other.floors) <= 1:
                return True
            else:
                return False
        else:
            return False


class Apartment(HouseScheme):

    def __init__(self, rooms, square, bathroom, floors, window_view):
        if isinstance(floors, int) and (1 <= floors <= 15) and
isinstance(window_view, str) and (
                window_view in ['N', 'S', 'W', 'E']):
            super().__init__(rooms, square, bathroom)
            self.floors = floors
            self.window_view = window_view
```

```
        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.bathroom, self.floors,
self.window_view)


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):
        return sum(element.square for element 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):
                super().append(element)

    def floor_view(self, floors, directions):
        ans = list(
            filter(lambda element: (floors[0] <= element.floors <=
floors[1]) and (element.window_view in directions),
                   self))
        for i in ans:
            print(str(i.window_view) + ": " + str(i.floors))
```