

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. 0383

Шквиря Е.В.

Преподаватель

Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Изучить устройство объектно-ориентированного подхода к программированию на языке Python и использование различных парадигм.

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 повторяться. Для реализации используйте функцию filter().

'''

Выполнение работы.

Порядок выполнения поставленной задачи:

1. Для реализации классов, отведённых под здания, был создан класс-родитель (*HouseScheme*) с минимальным набором необходимых полей, от которого в последствии наследовались все дочерние классы (*CountryHouse*, *Apartament*) и дополнялись своими полями и переопределёнными методами. Конструкторы классов содержат проверку на корректность переданных аргументов и в случае некорректности данных могут выбросить исключение *ValueError*.
2. Для группировки зданий были реализованы классы *CountryHouseList* и *ApartamentList*, которые наследуются от класса *list*. В них были переопределены некоторые методы и добавлены свои.

Иерархия классов:



3. В ходе работы были перегружены методы:

Класс object:

`__init__()` - конструктор класса

`__str__()` - приведение класса к типу *str*

`__eq__()` - оператор «==»

Класс list:

append() - добавить объект в конец списка

extend() - добавить несколько объектов в конец списка

Используемые в программе переменные:

cnt_rooms (cntRooms) — количество комнат

square_room (squareRoom) — площадь комнаты

comb_bath (combBath) — совмещённый санузел

cnt_floors (cntFloors) — количество этажей

square_land (squareLand) — площадь участка

floor — номер этажа

windows_dir (windowsDir) — сторона, на которую выходит окно

name — название здания

p_object — объект, который потенциально может быть добавлен в конец списка

sum — сумма площадей комнат

iterable — объект, из которого потенциально могут добавиться элементы в конец списка

floors — список с нижней и верхней границей допустимых этажей

directions — строка с допустимыми направлениями окон

low — нижняя граница допустимых этажей

up — верхняя граница допустимых этажей

it — объект-итератор, в котором хранятся отобранные элементы из ApartmentList

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	HouseScheme(1,2,False)		Исключение отсутствует
2.	HouseScheme (3, 4, "True")		Произошло исключение
3.	CountryHouse(1,2,True,4,5)		Исключение отсутствует
4.	CountryHouse(5,"4",True,4,"5")		Произошло исключение
5.	Apartament(1,2,True,4,5)		Произошло исключение
6.	Apartament(9,8,True,6,"S")		Исключение отсутствует
7.	CountryHouseList("abc")		Исключение отсутствует
8.	ApartamentList("abc")		Исключение отсутствует

Выводы.

Было изучено устройство объектно-ориентированного подхода к программированию на языке Python, методика использования наследования и полиморфизма. В основе использовалась императивная парадигма.

Разработана система классов для градостроительной компании, позволяющая задавать различные типы зданий и группировать их в списки. Для выполнения задачи использовался объектно-ориентированный подход, создание классов, переопределение методов и наследование.

Доп. вопросы.

3) Метод `__str__()` будет вызывать при попытке привести объект к типу `str`. Например, при попытке вывести этот элемент через функцию `print()`

4) Да, непереопределённые методы класса `list` будут работать для `CountryHouseList` и `ApartamentList`. В следствии того, что они не переопределены, они будут иметь реализацию, описанную в родительском классе, которая будет работать с полями, которые также описаны в родительском классе. Например, функция `__len__()` также будет считать количество элементов в списке, а `__eq__()` также будет проверять два списка на равенство объектов внутри них.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3.py

```
class HouseScheme:
    def __init__(self, cnt_rooms, square_room, comb_bath):
        if type(cnt_rooms) is not int or type(square_room) is not int or type(comb_bath) is not
bool or \
            cnt_rooms < 0 or square_room < 0:
            raise ValueError("Invalid value")
        self.cntRooms = cnt_rooms
        self.squareRoom = square_room
        self.combBath = comb_bath

class CountryHouse(HouseScheme):
    def __init__(self, cnt_rooms, square_room, comb_bath, cnt_floors, square_land):
        super().__init__(cnt_rooms, square_room, comb_bath)
        if type(cnt_floors) is not int or type(square_land) is not int or cnt_floors < 0 or
square_land < 0:
            raise ValueError("Invalid value")
        self.cntFloors = cnt_floors
        self.squareLand = square_land

    def __str__(self):
        return "Country House: Количество жилых комнат {}, Жилая площадь {}, " \
            "Совмещенный санузел {}, Количество этажей {}, " \
            "Площадь участка {}".format(self.cntRooms, self.squareRoom, self.combBath,
self.cntFloors,
            self.squareLand)

    def __eq__(self, other):
        return self.squareRoom == other.squareRoom and self.squareLand ==
other.squareLand and abs(
            self.cntFloors - other.cntFloors) <= 1
```

```

class Apartment(HouseScheme):
    def __init__(self, cnt_rooms, square_room, comb_bath, floor, windows_dir):
        super().__init__(cnt_rooms, square_room, comb_bath)
        if type(floor) is not int or type(windows_dir) is not str or floor < 1 or windows_dir not in
"NSWE" \
        or floor > 15:
            raise ValueError("Invalid value")
        self.floor = floor
        self.windowsDir = windows_dir

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {}, ' \
        'Совмещенный санузел {}, Этаж {}, Окна выходят на {}.' \
        .format(self.cntRooms, self.squareRoom, self.combBath, self.floor, self.windowsDir)

```

```

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

    def append(self, p_object):
        if type(p_object) is not CountryHouse:
            raise TypeError("Invalid type {}".format(type(p_object)))
        super().append(p_object)

    def total_square(self):
        sum = 0
        for item in self:
            sum += item.squareRoom
        return sum

```

```

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

```

```
def extend(self, iterable):
    for item in iterable:
        if type(item) is Apartment:
            self.append(item)

def floor_view(self, floors, directions):
    low, up = floors
    it = filter(lambda x: True if low <= x.floor <= up and x.windowsDir in directions else
False, self)
    for item in it:
        print("{}: {}".format(item.windowsDir, item.floor))
```