

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

Охотникова Г.С.

Преподаватель

Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Освоить парадигму объектно-ориентированного программирования.

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

"""

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().

'''

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.

ООП в Python:

Классы содержат атрибуты, которые подразделяются на поля и методы. Под методом понимают функцию, которая определена внутри класса.

Конструктор - это специальный метод, который нужен для создания объектов класса.

Функция filter():

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

Lambda-выражения:

Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций. Синтаксис определения следующий:
lambda аргумент1, аргумент2,..., аргументN : выражение

Выполнение работы.

Класс *HouseScheme()*. Не имеет родителя, имеет два класса-потомка *Apartment* и *CountryHouse*. В конструкторе инициализируются поля объекта класса *count_rooms*(количество жилых комнат), *square*(жилая площадь), *bathroom*(совмещенный санузел (значениями могут быть или *False*, или *True*)). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Класс *CountryHouse(HouseScheme)*. Наследник класса *HouseScheme*, не является родителем. В конструкторе наследуются поля объекта класса *HouseScheme* - *count_rooms* , *square*, *bathroom* и инициализируются другие поля – *count_floors*(количество этажей), *square2*(площадь участка). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Затем происходит переопределение метода *__str__(self)*, который возвращает строку заданного формата, и метода *__eq__(self, object2)*, который возвращает *True*, если два объекта класса, переданные в метод, равны, и *False* в противном случае.

Класс *Apartment(HouseScheme)*. Потомок класса *HouseScheme*, не является родителем. В конструкторе наследуются поля объекта класса *HouseScheme* - *count_rooms* , *square*, *bathroom* и инициализируются другие поля – *floor*(этаж (может быть число от 1 до 15)), *window*(куда выходят окна (значением может быть одна из строк: N, S, W, E)). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Затем переопределяется метод *__str__(self)*, который возвращает строку заданного формата.

Класс *CountryHouseList(list)*. Потомок класса *list*, не является родителем. В конструкторе инициализируется поле объекта класса – *name*

(полно класса присваивается аргумент-строки *name*). Затем происходит переопределение метода *append(self, p_object)*, в котором осуществляется проверка переданного аргумента. Если переданный в метод аргумент *p_object* удовлетворяет заданным условиям, то элемент добавляется в список, иначе выбрасывается исключение *TypeError* с текстом: *'Invalid type <тип_объекта p_object>'*. Также происходит переопределение метода *total_square(self)*, в котором считается и возвращается общая жилая площадь текущего объекта класса.

Класс *ApartmentList(list)*. Потомок класса *list*, не является родителем. В конструкторе инициализируется поле объекта класса – *name*(полно класса присваивается аргумент-строки *name*). Затем происходит переопределение метода списка - *extend(self, iterable)*, в котором осуществляется проверка, элемента *iterable*. Если элемент *iterable* - объект класса *Apartment*, он добавляется в список. Также переопределяется метод *floor_view(self, floors, directions)*. Метод в качестве параметров получает диапазон возможных этажей в виде списка и список направлений из ('N', 'S', 'W', 'E'). Метод выводит квартиры, удовлетворяющие заданным условиям, окна которых выходят в одном из переданных направлений, преобразуя их в строку заданного формата.

1. Иерархия описанных классов.

CountryHouse(потомок) – *HouseScheme*(родитель)

Apartment(потомок) – *HouseScheme*(родитель)

CountryHouseList(потомок) - *list*(родитель)

ApartmentList(потомок) - *list*(родитель)

2. Методы, которые были переопределены:

```
def __init__(self, );
```

```
def __str__(self);
```

```
def __eq__(self, object2);
```

```
def append(self, p_object);
```

```
def extend(self, iterable).
```

3. Метод *__str__()* будет вызван:

При вызове функции *str()* - приведении к типу “строка” в явном виде, или неявном, как, например, при вызове функции *print()*.

4. Будут ли работать непереопределенные методы класса *list* для *CountryHouseList* и *ApartmentList*? Объясните почему и приведите примеры.

Будут, но если не переопределять - будут работать в их базовом формате, как обычные функции класса *list*, ведь он является родителем классов *CountryHouseList* и *ApartmentList*.

Пример: метод *list.clear()*, если его не переопределить, будет очищен нынешний список, являющийся объектом класса *CountryHouseList* или *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.	<code>print(house1)</code>	Country House: Количество жилых комнат 5, Жилая площадь 55, Совмещенный санузел True, Количество этажей 2, Площадь участка 900.	Программа работает правильно.
2.	<code>print(apartment1)</code>	Apartment: Количество жилых комнат 3, Жилая площадь 70, Совмещенный санузел False, Этаж 9, Окна выходят на S.	Программа работает правильно.
3.	<code>list1 = CountryHouseList('Country') list1.append(house1)</code>	Country House: Количество жилых комнат 5, Жилая площадь 55, Совмещенный санузел	Программа работает правильно.

	print(list1)	True, Количество этажей 2, Площадь участка 900. Apartment: Количество жилых комнат 3, Жилая площадь 70, Совмещенный санузел False, Этаж 9, Окна выходят на S. [<__main__.CountryHouse object at 0x000002820B350190>]	
...			

Выводы.

Была изучена объектно-ориентированная парадигма.

Разработана программа, описывающая классы и их методы, были предусмотрены случаи возникновения исключений.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

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

class CountryHouse(HouseScheme):
    def __init__(self, count_rooms, square, bathroom,
count_floors, square2):
        super().__init__(count_rooms, square, bathroom)
        if count_floors > 0 and square2 > 0:
            self.count_floors = count_floors
            self.square2 = square2
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая
площадь {}, Совмещенный санузел {}, Количество этажей {}, Площадь
участка {}'.format(
            self.count_rooms, self.square, self.bathroom,
self.count_floors, self.square2)

    def __eq__(self, other):
        if self.square == other.square and self.square2 ==
other.square2 and abs(
            self.count_floors - other.count_floors) <= 1:
            return True
        else:
            return False

class Apartment(HouseScheme):
    def __init__(self, count_rooms, square, bathroom, floor,
window):
        super().__init__(count_rooms, square, bathroom)
        if floor >= 1 and floor <= 15 and window in ['N', 'S',
'W', 'E']:
            self.floor = floor
            self.window = window
        else:
            raise ValueError('Invalid value')

    def __str__(self):
```

```

        return 'Apartment: Количество жилых комнат {}, Жилая
площадь {}, Совмещенный санузел {}, Этаж {}, Окна выходят на
{}'.format(
self.count_rooms, self.square, self.bathroom, self.floor,
self.window)

```

```

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

    def append(self, p_object):
        if isinstance(type(p_object), CountryHouse):
            super().append(p_object)
        else:
            raise TypeError(f'Invalid type {type(p_object)}')

    def total_square(self):
        __total_area = 0
        for i in self:
            __total_area += i.square
        return __total_area

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

    def extend(self, iterable):
        super().extend(filter(lambda x: type(x) == Apartment,
iterable))

    def floor_view(self, floors, directions):
        checking = list(filter(lambda i: (i.window in directions)
and (i.floor in list(range(floors[0], floors[1] + 1))), self))
        for i in checking:
            print('{}: {}'.format(i.window, i.floor))

```