МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Информатика»

Тема: Парадигмы программирования

Студент гр. 0382 _____ Ильин Д.А.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург

2020

Цель работы.

Освоение основных принципов работы с классами, изучение парадигм 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 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 повторятся. Для реализации используйте функцию filter().

'''

Выполнение работы.

В программе имеется 5 классов:

- HouseScheme — не является наследником
- CountryHouse — наследник класса HouseScheme
- Apartment — наследник класса HouseScheme
- CountryHouseList — наследник класса list
- ApartmentList — наследник класса list

1)Класс HouseScheme:

Поля объекта класса инициализированные в конструкторе:

- self.room — количество комнат
- self.size — жилая площадь
- self.san — совмещённый санузел или нет

Также идёт проверка условий- переданное значение жилой площади больше равно нуля, тип переданного значения в self.san является bool. Если данные подходят под условия, то соответствующие поля инициализируются, иначе принудительно вызывается ошибка ValueError, с параметром Invalid value.

2)Класс CountryHouse:

Поля объекта класса инициализированные в конструкторе:

- self.room — количество комнат
- self.size — жилая площадь
- self.san — совмещённый санузел
- self.height — количество этажей
- self.space — площадь участка

Первые 3 поля инициализируются при помощи родительского конструктора.

Переопределяется метод __str__, при приведении объекта класса к строковому типу теперь будет возвращена строка:

Country House: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Количество этажей <количество этажей>, Площадь участка <площадь участка>.

Переопределяется метод __eq__, проверяющий два элемента на эквивалентность. Метод возвращает True если жилая площадь двух объектов равна и количество этажей отличается не более, чем на 1, иначе возвращает False.

3)Класс Apartment:

Поля объекта класса инициализированные в конструкторе:

- self.room — количество комнат
- self.size — жилая площадь
- self.san — совмещённый санузел
- self.floor — этаж
- self.side — куда выходят окна

Первые 3 поля инициализируются при помощи родительского конструктора.

Также идёт проверка условий — переданное значение этажа от 1 до 15 и тип переданного значения в self.side является одним из следующих значений: 'N', 'S', 'W', 'E'. Если данные подходят под условия, то соответствующие поля инициализируются, иначе принудительно вызывается ошибка ValueError, с параметром Invalid value.

Переопределяется метод __str__, при приведении объекта класса к строковому типу теперь будет возвращена строка:

Apartment: Количество жилых комнат <количество жилых комнат>, Жилая площадь <жилая площадь>, Совмещенный санузел <совмещенный санузел>, Этаж <этаж>, Окна выходят на <куда выходят окна>.

4)Класс CountryHouseList(list):

При создании объекта класса вызывается конструктор родительского класса list, а также инициализируется поле self.name

Переопределяется метод append следующим образом- если поданный объект класса CountryHouse, вызывается метод append родительского класса,

иначе принудительно вызывается ошибка TypeError, с параметром „Invalid type <класс поданного объекта>.“

Создаётся метод total_square который считает и возвращает общую площадь всех объектов, которые лежат в self.

5)Класс ApartmentList(list):

При создании объекта класса вызывается конструктор родительского класса list, а также инициализируется поле self.name

Переопределяется метод extend следующим образом- тип каждого элемента поданного списка проверятся, из всех элементов типа Apartment формируется новый список, с которым вызывается метод extend родительского класса.

Создаётся метод floor_view находит все подходяшие элементы из self, и выводит на экран строки типа: <Направление_1>: <этаж_1>

Иерархия описанных классов:

HouseScheme — родительский класс для CountryHouse и Apartment

list — родительский класс для CountryHouseList и ApartmentList

Методы которые были переопределены:

HouseScheme — __init__()

CountryHouse — __init__(), __str__(), __eq__()

Apartment — __init__(), __str__()

CountryHouseList — __init__(), append()

ApartmentList — __init__(), extend)(

Метод __str__() будет вызван при приведении объекта класса к строковому типу, к примеру при вызове функции str().

Не переопределённые методы родительского класса будут работать и на дочерних классах по определению. В таком случае и не переопределённые

методы класса list будут работать в классах CountryHouseList и ApartmentList.

К примеру метод append() будет работать с объектами класса ApartmentList, и будет к этому объекту добавлять элемент любого типа.

Тестирование.

Для тестирования используем код, приведённый ниже.

```
a = Apartment(2, 22, True, 2, 'N')
a2 = Apartment(2, 22, True, 2, 'W')
a3 = Apartment(2, 22, True, 7, 'N')
a4 = Apartment(2, 22, True, 1, 'N')
a5 = Apartment(2, 22, True, 15, 'N')

b = CountryHouse(2,11, True, 10, 11)
b1 = CountryHouse(2,10, True, 10, 11)
b2 = CountryHouse(2,10, True, 10, 11)
b3 = CountryHouse(2,10, True, 10, 11)
b4 = CountryHouse(2,10, True, 10, 11)

s1 = CountryHouseList("Русь")
s1.append(b)
s1.append(b1)
s1.append(b2)
s1.append(b3)
s1.append(b4)
print(s1.total_square())

s = ApartmentList("Русь")
s.extend([a,a2,a3,a4,a5])
print(s.floor_view([2,7], ['N', 'W']))
```

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.		51 W: 2 N: 7 N: 2 None	Программа работает корректно

Выводы.

Была написана система классов для градостроительной компании, а также отчёт, на который было потрачено времени сильно больше нежели на написание самой программы.

Были изучены основные возможности классов и варианты работы с ними, были изучены функции map() и filter(), а также способы работы с lambda выражениями.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

```
class HouseScheme:
    def __init__(self, new_room, new_size, new_san):
        if ((new_size >= 0) and (type(new_san) ==
bool)):
            self.room = new_room
            self.size = new_size
            self.san = new_san
        else:
            raise ValueError('Invalid value')

class CountryHouse(HouseScheme):
    def __init__(self, new_room, new_size, new_san,
new_height,
                 new_space):
        super().__init__(new_room, new_size,
new_san)
        self.height = new_height
        self.space = new_space

    def __str__(self):
        return f'Country House: Количество жилых
комнат {self.room}, Жилая площадь {self.size},
Совмещенный санузел {self.san}, Количество этажей
{self.height}, Площадь участка {self.space}.'

    def __eq__(self, other):
        if (self.size == other.size) and
(self.height -1 <= other.height <= self.height +1):
            return True
        else:
            return False

class Apartment(HouseScheme):
    def __init__(self, new_room, new_size, new_san,
new_floor,
                 new_side):
        if ((1 <= new_floor <= 15) and (new_side in
['N', 'S', 'W', 'E'])):
            super().__init__(new_room, new_size,
new_san)
            self.floor = new_floor
            self.side = new_side
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return f"Apartment: Количество жилых комнат
```

```
{self.room}, Жилая площадь {self.size}, Совмещенный
санузел {self.san}, Этаж {self.floor}, Окна выходят
на                                          {self.side}."

class                          CountryHouseList(list):

                    def      __init__(self,      name):
                                    super().__init__()
                                 self.name    =    name

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

                           def      total_square(self):
                                          __a     =     0
                                for    i    in    self:
                                      __a   +=   i.size
                                         return      __a

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):
        norm_hight = lambda x: True if (floors[0] <=
x.floor)  and  (x.floor  <=  floors[1])  else  False
           norm_side = lambda x: True if x.side in
directions                  else                False
        norm_Apartment = list(set(filter(norm_hight,
self))      &      set(filter(norm_side,      self)))
                                    endstr    =    []
                       for   i   in   norm_Apartment:
                endstr.append(f'{i.side}: {i.floor}')
        print('\n'.join(endstr))
```