МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического Обеспечения и Применения ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Информатика»
Тема: Парадигмы программирования

Студент гр. 0382 _____ Павлов С. Р.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург
2020

Цель работы.

Изучить основные парадигмы программирования, используя язык программирования Python.

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 у бедиться, 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().

'''

В отчете укажите:

1. Иерархию описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будет вызван метод __str__().
4. Будут ли работать непереопределенные методы класса list для CountryHouseList и ApartmentList? Объясните почему и приведите примеры.

Основыные теоритеческие положения.

В данной лабораторной работе были использованы следующие конструкции языка python:

- *class Myclass(Baseclass)* -объявление класса с именем *Myclass*, наследующегося от класса *Baseclass*.
- *super().some_parent_method()* - вызов метода родительского класса.
- *def __init__()* - магический метод. Вызывается при создании нового объекта класса.

- *def __str__()* - магический метод. Вызывается при попытке получить строковое представление объекта.
- *lambda x*: ... – создание безымянной функции.
- *filter(function, iterable)* - возвращает итерируемый объект, состоящий из тех элементов *iterable*, для которых функция *function* вернет значение *True*.
- *map(function, iterable)* - возвращает итерируемый объект, где каждый элемент – это результат работы функции с соответствующим элементом *iterable*.

Выполнение работы.

В ходе работы была создана следующая иерархия классов: Классы *CountryHouse* и *Apartment*, наследующиеся от класса *HouseScheme* и классы *CountryHouseList* и *ApartmentList*, наследующиеся от класса *list*.

- Класс *CountryHouse*.

Для создания этого класса был переопределен ряд следующих методов: __init__(), __str__(), __eq__() .

Переопределенный метод *__init__()* имеет шесть параметров, один из которых – ссылка на объект. В данном методе сначала вызывается одноименный метод родительского класса, после чего выполняется проверка на неотрицательность аргументов *land_area* и *floors_number*. Если условие ложно, то соответствующим атрибутам присваиваются значения параметров. В противном случае возбуждается исключение *ValueError()*.

В методе *__str__()* с помощью метода строк *format()* возвращается строка, удовлетворяющая условию задания. В переопределенном методе __eq__() выполняется проверка на равенство значений атрибутов *square* и *land_area* объекта класса и переданного объекта. Также выполняется проверка на то, что разница между значениями атрибута *floors_number* , через

переменную symmary не превышает двух. Если условие выполнено возвращается значение True. Иначе – значение False.

- Класс *Apartment*.

Для создания этого класса переопределен ряд следующих методов: __init__(), __str__().

Переопределенный метод *__init__()* принимает на вход шесть аргументов. После вызова одноименного метода родительского класса, выполняется проверка на то, что значение параметра *floor_number* лежит в диапозоне от 1 до 16 и значение параметра *windows_direct* присутствует в списке допустимых значений. Если данные некорректны, возбуждается исключение *ValueError()*. В противном случае атрибутам класса *Apartment* присваиваются значения одноименных параметров.

Переопределенный метод *__str__()* действует аналогично одноименному методу класса *CountryHouse*.

- Класс *CountryHouseList*.

Для создания этого класса объявлены и переопределены следующие методы: __init__(), append(), total_square().

Переопределенный метод *__init__()* принимает на вход два параметра: ссылку на объект и название. После вызова родительского метода *__init__()* создается атрибут с именем *name*, которому присваивается значение одноименного аргумента.

Метод *append* принимает на вход два аргумента: ссылку на объект и объект, который необходимо добавить. Далее выполняется проверка на то, что добавляемый объект является объектом класса *CountryHouse*. Если условие ложно, выбрасывается исключение *TypeError*. В противном случае объект добавляется в список с помощью родительского метода *append*.

Метод *total_square* суммирует значения атрибута *square* всех объектов, хранящихся в списке, данная сумма возвращается в качестве результата работы программы.

- Класс *ApartmentList.*

Для создания класса переопределены и объявлены следующие методы: __init__(), extend(), floor_view()

Метод *__init__()* идентичен одноименному методу класса *CountryHouseList*.

Метод *extend* помимо ссылки на объект, принимает на вход итерируемый объект. Полученный итерируемый объект присоединяется к списку с помощью функции родителя *extend*.

Метод *floor_view* принимает на вход ссылку на объект, допустимый диапозон значений для этажей и допустимые значения для атрибута *windows_direct*. Сначала с помощью функции *filter* извлекаются объекты списка, удовлетворяющие условию, после чего формируется строка и выводится на экран.

Не переопределённые методы класса *list* для классов *CountryHouseList* и *ApartmentList* работать будут, потому что *ApartmentList* и *CountryHouseList* являются наследниками класса *list*, поэтому все методы, присущие классу *list* будут также работать для всех объектов классов-наследников. Например не переопределённый метод *self.reverse()* развернет список.

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.	`a = CountryHouse(4, 80, True,1, 90)` `print(a)`	Country House: Количество жилых комнат 6, Жилая плоадь 110, Совмещенный санузел True, Количество этажей 1, Площадь участка 120.	Программа работает правильно
2.	`b = CountryHouse(7,111, True,1,120)` `print(a.__eq__(b))`	True	Программа работает правильно

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, rooms_number, square, bathroom):

        if square < 0 or type(bathroom) != bool:
            raise ValueError("Invalid value")
        else:
            self.rooms_number = rooms_number
            self.square = square
            self.bathroom = bathroom

class CountryHouse(HouseScheme):
          def __init__(self, rooms_number, square, bathroom,
floors_number, land_area):

        super().__init__(rooms_number, square, bathroom)

        if (land_area < 0) or (floors_number < 0):
            raise ValueError('Invalid value')

        else:
            self.floors_number = floors_number
            self.land_area = land_area

    def __str__(self):
        return 'Country House: Количество жилых комнат {0}, Жилая
площадь {1}, Совмещенный санузел {2}, Количество этажей {3}, Площадь
участка {4}.'.format(self.rooms_number, self.square, self.bathroom,
self.floors_number, self.land_area)


    def __eq__(self, obj):
        summary = abs(self.floors_number - obj.floors_number)
          if ((self.square == obj.square) and (self.land_area ==
obj.land_area) and (summary < 2)):
            return True
        return False

class Apartment(HouseScheme):
          def __init__(self, rooms_number, square, bathroom,
floor_number, windows_direct):

        super().__init__(rooms_number, square, bathroom)

        #Условие
             if (floor_number < 1) or (floor_number > 16) or
(windows_direct not in ('N', 'S', 'W', 'E')):
            raise ValueError('Invalid value')
        else:
            self.floor_number = floor_number
            self.windows_direct = windows_direct
```

```
        def __str__(self):
            return 'Apartment: Количество жилых комнат {0}, Жилая площадь {1}, Совмещенный санузел {2}, Этаж {3}, Окна выходят на {4}.'.format(self.rooms_number, self.square, self.bathroom, self.floor_number, self.windows_direct)


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

        def append(self, p_object):
            if (isinstance(p_object, CountryHouse)==0):
                raise TypeError('Invalid type {}'.format(type(p_object)))

            else:
                super().append(p_object)

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

    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, r, directs):
            apartments = list(filter(lambda x: x.floor_number in range(r[0], r[1] + 1) and x.windows_direct in directs, self.__iter__()))
            print('\n'.join(list(map(lambda x: x.windows_direct + ': ' + str(x.floor_number), apartments))))
```