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 _____ Михайлова О.Д.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Рассмотреть понятия парадигм программирования и освоить объектно-ориентированное программирование в 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 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().

'''

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.

Основные теоретические положения.

Парадигма программирования – это подход к программированию, описанный совокупностью идей и понятий, определяющих стиль написания компьютерных программ.

Итератор – это специальный объект, который делает проще переходы по элементам другого объекта. Итератор – это своего рода перечислитель для

определенного объекта (например, списка, строки, словаря), который позволяет перейти к следующему элементу этого объекта, либо бросает исключение, если элементов больше нет.

Итерируемый объект – объект, по которому можно итерироваться (то есть который можно обходить в цикле, например, цикле for).

Лямбда-выражения – это специальный элемент синтаксиса для создания анонимных (т.е. без имени) функций по месту их использования. Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций.

Синтаксис:

lambda аргумент1, аргумент2, ..., аргументN : выражение

Функция filter:

- Функция filter(<функция>, <объект>) возвращает объект-итератор, состоящий из тех элементов итерируемого объекта <объект>, для которых <функция> является истиной.
- Функция <функция> применяется для каждого элемента итерируемого объекта <объект>.
- После того, как к итератору произошло обращение, из него извлекаются элементы.
- Чтобы воспользоваться результатами работы функции filter и после обращения к объекту-итератору, нужно обернуть вызов функции filter в функцию list(): list(filter(..., …)) , например:

print(list(filter(check_num, number_list)))

Объектно-ориентированное программирование:

Объект – конкретная сущность предметной области.

Класс – это тип объекта.

Метод – функция, которая определена внутри класса.

Поле – это переменная, которая определена внутри класса.

Конструктор – это специальный метод, который нужен для создания объектов класса. Синтаксис конструктора:

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

<Тело_конструктора>

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.

Под инкапсуляцией часто понимают сокрытие внутренней реализации от пользователя. В языке Python этот механизм лишь указывает, что атрибут не должен быть изменен.

Когда говорят о полиморфизме в контексте ООП, обычно говорят о переопределении методов.

Исключения – это специальный класс объектов в языке Python. Исключения предназначены для управления поведением программой, когда возникает ошибка, или, другими словами, для управления теми участками программного кода, где может возникнуть ошибка.

Выполнение работы.

Ход решения:

В программе реализованы следующие классы:

1) Класс HouseScheme. В конструкторе __init__() инициализируются поля:

- self.count_rooms – количество жилых комнат;
- self.square – площадь;
- self.combined_bath – совмещенный санузел.

Если при создании экземпляра класса переданные в конструктор параметры не удовлетворяют требованиям, что площадь не может быть отрицательной и поле combined_bath имеет тип bool, то с помощью оператора raise создается и выбрасывается исключение ValueError с текстом ‘Invalid value’.

1) Класс CountryHouse. Наследник класса HouseScheme. В конструкторе класса наследуются поля класса-родителя и инициализируются поля:

- self.count_floors – количество этажей;
- self.land_area – площадь участка.

При создании экземпляра так же происходит проверка, что переданные в конструктор параметры удовлетворяют требованиям.

Далее переопределяются методы:

- __str__() – преобразует к строке определенного вида;
- __eq__() – возвращает True, если два объекта класса равны, и False иначе.

1) Класс Apartment. Наследник класса HouseScheme. В конструкторе класса наследуются поля класса-родителя и инициализируются поля:

- self.floor – этаж;
- self.window – куда выходят окна.

Если при создании экземпляра класса переданные в конструктор параметры не удовлетворяют требованиям, что поле floor – это число от 1 до 15, а поле window – принимает одно из значений ‘N’, ‘S’, ‘W’, ‘E’, то с помощью оператора raise создается и выбрасывается исключение ValueError с текстом ‘Invalid value’.

Далее переопределяются методы:

- __str__() – преобразует к строке определенного вида.

1) Класс CountryHouseList. Наследник класса list. В конструкторе класса инициализируются поле self.name.

Далее переопределяется метод append(p_object) и определяется метод total_square(). В методе append проверяется, является ли p_object деревенским домом. Если да, то элемент добавляется в список. Если нет, то выбрасывается исключение TypeError с текстом ‘Invalid type <тип объекта p_object>’. В методе total_square считается общая жилая площадь.

2) Класс ApartmentList. Наследник класса list. В конструкторе класса инициализируется поле self.name.

Далее переопределяется метод extend(iterable) и определяется метод floor_view(floor, direcrions). В методе extend проверяется, является ли iterable объектом класса Apartment. Если да, то элемент добавляется в список. Метод floor_view в качестве параметров получает диапазон возможных этажей в виде списка floor и список направлений, куда могут выходить окна, directions. Метод выводит квартиры в виде строки определенного типа, удовлетворяющие условиям, что их этаж находится в заданном диапазоне возможных этажей и окна выходят в одном из заданных направлений.

Иерархия описанных классов:

1. Класс-родитель: HouseScheme
   Классы-потомки: CountryHouse, Apartment
2. Класс-родитель: list
   Классы-потомки: CountryHouseList, ApartmentList

Методы, которые были переопределены:

- __init__(self, …);
- __str__(self);
- __eq__(self, other);
- append(self, p_object);
- extend(self, iterable).

Метод __str__() будет вызван в том случае, когда объект приводит к строковому типу. Например, при вызове функции str().

Не переопределенные методы класса list будут работать для CountryHouseList и ApartmentList, так как они являются наследниками класса list, и все методы, определенные для этого класса, будут определены и для его потомков. Например, не переопределенный метод self.revers() развернет текущий список, являющийся объектом класса CountryHouseList или ApartmentList.

Разработанный программный код см. в приложении А.

Тестирование.

Экземпляры классов, созданные для проверки:

house1 = CountryHouse(5, 300, True, 1, 500)

house2 = CountryHouse(7, 300, False, 2, 500)

apartment1 = Apartment(3, 100, False, 5, 'N')

apartment2 = Apartment(4, 200, True, 2, 'S')

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	print(house1)	Country House: Количество жилых комнат 5, Жилая площадь 300, Совмещенный санузел True, Количество этажей 1, Площадь участка 500.	Проверка метода __str__(). Результат верный.
2.	print(house1 == house2)	True	Проверка метода __eq__().

			Результат верный.
3.	print(apartment1)	Apartment: Количество жилых комнат 3, Жилая площадь 100, Совмещенный санузел False, Этаж 5, Окна выходят на N.	Проверка метода __str__(). Результат верный.
4.	list1 = CountryHouseList('house') list1.append(house1) list1.append(house2) print(list1)	[<__main__.CountryHouse object at 0x7faddd2df340>, <__main__.CountryHouse object at 0x7faddd2df400>]	Проверка метода append(). Результат верный.
5.	print(list1.total_square())	600	Проверка метода total_square(). Результат верный.
6.	list2 = ApartmentList('apartment') list2.extend([apartment1, 'cgh', apartment2]) print(list2)	[<__main__.Apartment object at 0x7fd1995e84c0>, <__main__.Apartment object at 0x7fd1995e8520>]	Проверка метода extend(). Результат верный.
7.	print(list2.floor_view([1, 6],['N']))	N: 5	Проверка метода floor_view(). Результат верный.

Выводы.

Были рассмотрены понятия парадигм программирования и освоено объектно-ориентированное программирование в Python.

Разработана программа, в которая реализована система классов для градостроительной компании с помощью объектно-ориентированного программирования. Также применялись исключения, функции lambda и filter.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: inf_lb_3.py

```
class HouseScheme:

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

class CountryHouse(HouseScheme):

        def __init__(self, count_rooms, square, combined_bath, count_floors, land_area):
        super().__init__(count_rooms, square, combined_bath)
        self.count_floors = count_floors
        self.land_area = land_area

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

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

class Apartment(HouseScheme):

    def __init__(self, count_rooms, square, combined_bath, floor, window):
        super().__init__(count_rooms, square, combined_bath)
         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.combined_bath, self.floor, self.window)


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

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

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

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

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

        def floor_view(self, floors, directions):
              check = list(filter(lambda x: x.floor >= floors[0] and
x.floor <= floors[1] and x.window in directions, self))
            for i in check:
           print('{}: {}'.format(i.window, i.floor))
```