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. 1304 _____ Шаврин А.П.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Научиться основам работы с ООП, исключениями и элементами функционального программирования на языке Python 3.

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().

'''

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.

ООП, наследование, переопределение родительских методов, *filter()*

Обработчик исключений try-except-else-finally

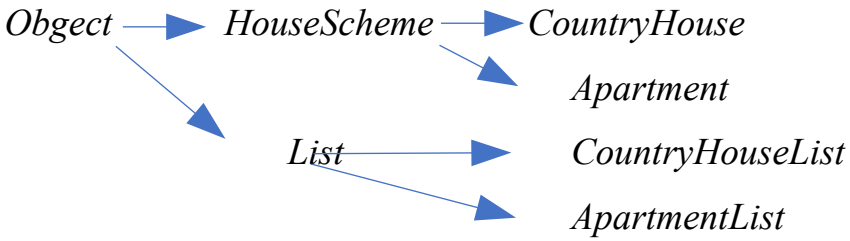
Исключения, с которыми вы уже, возможно,
столкнулись: *TypeError*, *ValueError*, *ZeroDivisionError* и т.д.
В языке Python выстроена иерархия исключений на основе ОО-подхода (см. полезные ссылки).
Кратко о блоках обработки исключительных ситуаций:

- В блок ***try*** помещают код, который может вызвать исключительную ситуацию (потенциально опасный код).
- В блок ***except*** помещают код для обработки исключительной ситуации. Для одного **try**-блока может быть несколько ***except***-блоков.

- В блок ***finally*** помещают код, который должен выполниться в любом случае, вне зависимости от того, произошла исключительная ситуация или нет. Блок ***finally*** может быть только один.
- В блок ***else*** помещают код, который должен выполниться, если в ***try***-блоке не случилось исключительной ситуации (***else***-блок выполняется в случае, если утверждение "Исключительная ситуация произошла" ложно).
  Блок ***else*** может быть только один.

Выполнение работы.

1. Иерархия классов Порядок наследования описанных классов представлен ниже.



2. Переопределенные методы.
   - *CountryHouse*:
     - *__init__* - принимает все параметры, необходимые для конструктора наследуемого класса *HouseScheme,* а также необходимые параметры для создания собственных полей, выдавая исключение, если параметры не соотвествуют заданным в задании условиям.
     - *__str__* - возвращает информацию об объекте в соответсвии с заданием.
     - *__eq__* - возвращает результат сравнения объектов типа *CountryHouse*.
   - *Appartment*:
     - *__init__* - принимает все параметры, необходимые для конструктора наследуемого класса *HouseScheme,* а также необходимые параметры для создания собственных полей, выдавая исключение, если параметры не соотвествуют заданным в задании условиям.

- *__str__* - возвращает информацию об объекте в соответсвии с заданием.

- *CountryHouseList*:
  - *append* — добавляет только объекты типа *CountryHouse* в список, в другом случае выдает исключение.

- *ApartmentList*:
  - *extend* — добавляет в конец списка только объекты типа *Apartment*.

1. В каких случаях будет вызван метод *__str__()*.

   Метод строкового вывода вызывается тогда, когда объект явно преобразовывается к типу *str*, например *str(a), f"{a}", print(a)* и т.д.

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

Полиморфизм в Python позволяет работать и с пользовательскими данными, поэтому непереопределенные методы будут работать, (кроме метода *sort()*, поскольку без дополнительно переопределенного метода *__eq__* у программы нет информации о том, каким образом сравнивать пользовательские объекты).

Примеры:

- Для *CountryHouseList*

- Код:

```
ch1 = CountryHouse(3, 60, True, 1, 100)
ch2 = CountryHouse(6, 120, True, 2, 200)
ch3 = CountryHouse(9, 180, False, 3, 300)
lst = [ch1, ch2, ch3]
chl = CountryHouseList('Test')
for house in lst:
    chl.append(house)
print(chl)
chl.reverse()
```

print(chl)

- Вывод:

[<__main__.CountryHouse object at 0x7f8287df17b0>,
<__main__.CountryHouse object at 0x7f8287df17e0>,
<__main__.CountryHouse object at 0x7f8287df1840>]

[<__main__.CountryHouse object at 0x7f8287df1840>,
<__main__.CountryHouse object at 0x7f8287df17e0>,
<__main__.CountryHouse object at 0x7f8287df17b0>]

- Для *ApartmentList*

- Код:

a1 = Apartment(3, 60, True, 5, 'S')

a2 = Apartment(2, 40, False, 1, 'N')

a3 = Apartment(1, 14, True, 15, 'W')

lst = [a1, a2, a3]

al = ApartmentList('Test')

for apart in lst:

al.append(apart)

print(al)

al.reverse()

print(al)

- Вывод:

[<__main__.Apartment object at 0x7f316e9897b0>,
<__main__.Apartment object at 0x7f316e9897e0>,
<__main__.Apartment object at 0x7f316e989840>]

[<__main__.Apartment object at 0x7f316e989840>,
<__main__.Apartment object at 0x7f316e9897e0>,
<__main__.Apartment object at 0x7f316e9897b0>]

Разработанный программный код см. в приложении А.

Выводы

Были изучены основы работы с ООП, исключениями и элементами функционального программирования на языке Python 3, а также разработана программа по созданию классов, определению полей внутри их конструкторов, созданию методов, наследованию и переопределению родительских методов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла*: main.py*

```
class HouseScheme:
    def __init__(self, count_rooms, living_space, comb_bath):
            if (type(count_rooms) is int) and (living_space >= 1) and (type(comb_bath) is bool):
                self.count_rooms = count_rooms
                self.living_space = living_space
                self.comb_bath = comb_bath
            else:
                raise ValueError('Invalid value')


class CountryHouse(HouseScheme):
        def __init__(self, count_rooms, living_space, comb_bath, count_floors, land_area):
        super().__init__(count_rooms, living_space, comb_bath)
        if (count_floors >= 1) and (type(land_area) is int):
            self.count_floors = count_floors
            self.land_area = land_area
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        inf = f'Country House: Количество жилых комнат {self.count_rooms}, '\
              f'Жилая площадь {self.living_space}, '\
              f'Совмещенный санузел {self.comb_bath}, '\
              f'Количество этажей {self.count_floors}, '\
              f'Площадь участка {self.land_area}.'
```

```
        return inf

    def __eq__(self, other):
        if isinstance(other, CountryHouse):
            if (self.living_space == other.living_space) 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, living_space, comb_bath, floor, window_side):
        super().__init__(count_rooms, living_space, comb_bath)
        if (1 <= floor <= 15) and (window_side in ['N', 'S', 'W', 'E']):
            self.floor = floor
            self.window_side = window_side
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        inf = f'Apartment: Количество жилых комнат {self.count_rooms}, '\
            f'Жилая площадь {self.living_space}, '\
            f'Совмещенный санузел {self.comb_bath}, '\
            f'Этаж {self.floor}, '\
            f'Окна выходят на {self.window_side}.'
        return inf


class CountryHouseList(list):
```

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

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

    def total_square(self):
        total_living_space = 0
        for house in self:
            total_living_space += house.living_space
        return total_living_space


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

    def extend(self, iterable):
        super().extend(filter(lambda i: isinstance(i, Apartment), iterable))

    def floor_view(self, floors, directions):
        suitable_apart = list(filter(lambda apart: (floors[0] <= apart.floor <=
floors[1]) and (apart.window_side in directions), self))
        for apart in suitable_apart:
            print(f'{apart.window_side}: {apart.floor}')
```