МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Информатика»

Тема: «Парадигмы программирования».

Студент гр. 1304 _____ Байков Е.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы

Изучить основные понятия парадигм программирования. Освоить основные парадигмы на практике, 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 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

1. Иерархия классов:

Классы *Apartment* и *CountryHouse* являются наследниками класса *HouseScheme*; классы *CountryHouseList* и *ApartmentList* наследники класса list; классы *list* и *HouseScheme* наследники класса *object.*

2. Переопределение методов:

В классе *HouseScheme* был переопределен метод *__init__()*. В нем происходит проверка введенных пользователем значений на корректность. В

случае корректности введенных значений они записываются в поля класса, иначе генерируется исключение *ValueError*, которое выводит *InvalidValue*.

В классе CountryHouse переопределен метод *__init__()*. С помощью функции *super()* добавлены поля класса *HouseScheme*, а также два новых поля, для которых реализована проверка на корректность введенных данных. В случае некорректности генерируется исключение *ValueError*, которое выводит сообщение *InvalidValue*. Переопределен метод *__str__()*, возвращающий информацию об объекте в виде строки *f'Country House: Количество жилых комнат {self.rooms_count}, Жилая площадь {self.square}, Совмещенный санузел {self.combination_bathroom}, Количество этажей {self.floors_count}, Площадь участка {self.plot_area}.'*. Переопределен метод *__eq__()* для сравнения экземпляров класса. Если объект, с которым сравнивается экземпляр класса, является экземпляром данного класса, то функция возвращает *False*.

В классе *Apartment* переопределен метод *__init__()*. С помощью функции *super()*добавлены поля класса *HouseScheme*, а также два поля с проверкой корректности введенных значений. В случае некорректности значений генерируется исключение *ValueError*, которое выводит сообщение *InvalidValue.* Переопределен метод *__str__()*, который возвращает информацию об объекте в виде строки *f'Apartment: Количество жилых комнат {self.rooms_count}, Жилая площадь {self.square}, Совмещенный санузел {self.combination_bathroom}, Этаж {self.floor}, Окна выходят на {self.windows_side}.'*

В классе *CountryHouseList* переопределен метод *__init__()* для записи значения в поле *name*. Переопределен метод *append()*, который добавляет в список только объекты типа *CountryHouse*, иначе генерируется исключение *TypeError*. Определен метод *total_square()*, считающий жилую площадь всех элементов списка и возвращает ее.

В классе *ApartmentList* переопределен метод *__init__(),* для записи значения в поле *name*. Переопределен метод *extend()*, который добавляет в

список только элементы, которые являются объектами класса *Apartment*. Определение является ли объект экземпляром класса *Apartment* происходит с помощью функции *filter*, *lambda*-функции и функции *isinstane.* Определен метод *floor_view()*, принимающий на вход диапазон этажей и список направлений, а после выводящий строки вида *f'{направление}: {этаж}'.*

3. Метод *__str__()* будет вызван тогда, когда нужно получить строковое представление объекта или вывести объект в виде строки.

4. Неопределенные классы метода *list* будут работать для классов *CountryHouseList* и *ApartmentList*, так как данные классы наследуются от класса *list*.

Пусть будут созданы экземпляры классов каждого из списков:

```
list_CH = CountryHouseList('New Country Houses')
list_A = ApartmentList('New Apartment')
```

А также объекты классов *CountryHouse*, *Apartments* и *int*:

```
obj_CH = CountryHouse(2, 10, True, 2, 20)
obj_A = Apartment(2, 40, True, 4, 'N')
obj_int = 2
```

С помощью метода *count* проверяется количество вхождений *obj_A* в *list_CH,* в который добавлен объект *obj_CH.* После в *list_A* добавляются все экземпляры, созданные ранее, а затем с помощью цикла *for* выводятся на экран.

```
list_CH.append(obj_CH)
list_A.append(obj_CH)
list_A.append(obj_A)
list_A.append(obj_int)
print(list_CH.count(obj_A))
print(list_A)
for elem in list_A:
    print(elem)
```

Вывод на экран:

0

[<__main__.CountryHouse object at 0x00000246EB5243A0>, <__main__.Apartment object at 0x00000246EB524310>, 2]

Country House: Количество жилых комнат 2, Жилая площадь 10, Совмещенный санузел True, Количество этажей 2, Площадь участка 20.

Apartment: Количество жилых комнат 2, Жилая площадь 40, Совмещенный санузел True, Этаж 4, Окна выходят на N.

2

Выводы

В ходе выполнения лабораторной работы были изучены основные понятия парадигм программирования. Была разработана программа, содержащая в себе систему классов для градостроительной компании.

Был описан базов класс *HouseScheme* и классы *CountryHouse* и *Apartment*, которые наследовались от *HouseScheme.* Также были описаны классы *CountryHouseList* и *ApartmentList*, которые наследовались от *list.*

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Baykov_Egor_lb3.py

```
class HouseScheme:
    def __init__(self, rooms_count, square, comb_bath) -> None:
        if type(rooms_count) == int and type(square) == int and square
>= 0 and type(comb_bath) == bool:
            self.rooms_count = rooms_count
            self.square = square
            self.combination_bathroom = comb_bath
        else:
            raise(ValueError('Invalid value'))

class CountryHouse(HouseScheme):
    def __init__(self, rooms_count, square, comb_bath, floors_count,
plot_area) -> None:
        super().__init__(rooms_count, square, comb_bath)
        if type(floors_count) == int and type(plot_area) == int:
            self.floors_count = floors_count
            self.plot_area = plot_area
        else:
            raise(ValueError('Invalid value'))

    def __str__(self) -> str:
        return f'Country House: Количество жилых комнат
{self.rooms_count}, Жилая площадь {self.square}, Совмещенный санузел
{self.combination_bathroom}, Количество этажей {self.floors_count},
Площадь участка {self.plot_area}.'

    def __eq__(self, house) -> bool:
        if isinstance(house, CountryHouse):
            if self.square == house.square and self.plot_area ==
house.plot_area and abs(self.floors_count - house.floors_count) <= 1:
                return True
            else:
                return False
        else: return False

class Apartment(HouseScheme):
    def __init__(self, rooms_count, square, comb_bath, floor,
windows_side) -> None:
        super().__init__(rooms_count, square, comb_bath)
        if type(floor) == int and 1 <= floor <= 15 and windows_side in
['N', 'S', 'W', 'E']:
            self.floor = floor
            self.windows_side = windows_side
        else:
            raise(ValueError('Invalid value'))

    def __str__(self) -> str:
        return f'Apartment: Количество жилых комнат {self.rooms_count},
Жилая площадь {self.square}, Совмещенный санузел
```

```
{self.combination_bathroom}, Этаж {self.floor}, Окна выходят на
{self.windows_side}.'

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

    def append(self, p_object) -> None:
        if isinstance(p_object, CountryHouse):
            super().append(p_object)
        else:
            raise(TypeError(f'Invalid type {type(p_object)}'))

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

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

    def extend(self, __iterable) -> None:
        super().extend(list(filter(lambda obj: isinstance(obj,
Apartment), __iterable)))

    def floor_view(self, floors, directions) -> str:
        out_floors = list(filter(lambda obj: floors[0] <= obj.floor <=
floors[1], self))
        for some_floor in out_floors:
            if some_floor.windows_side in directions:
                print(f'{some_floor.windows_side}: {some_floor.floor}')
```