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

# Отчет

**по лабораторной работе №3**
**по дисциплине «Введение в информационные технологии»**
**Тема: « Система классов для градостроительной компании»**

Студент гр. 9383 _____ Корсунов А.А.

Преподаватель _____ Розмачева Н.В.

Санкт-Петербург
2019

**Цель работы.**

Научиться работать с объектно-ориентированным программированием. Познакомиться с классами, научиться создаваться собственные классы и их наследников. Научиться переопределять методы классов.

**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.

**Выполнение работы.**

1. **Иерархия классов.**

   object << HouseScheme << CountryHouse

   object << HouseScheme << Apartment

   object << list << CountrtyHouseList

   object  << list << ApartmentList

2. **Переопределение методов.**

   __init__ - конструктор класса

   __str__ - переопределение метода __str__ для требуемого по заданию вывода

   __eq__ - переопределение метода __eq__ для правильного сравнения объектов класса

   append - переопределение метода в классе CounrtyHouseList(list)  для добавления объектов класса CountryHouse

   extend -  переопределение метода в классе ApartmentList(list) для добавление объектов класса Apartment.

3. Метод __str__ будет вызван в случаях попытки представления класса в строковом виде (Н-р, через print()).
4. Неопределенные методы классов CountryHouseList и ApartmentList будут работать, потому как являются наследниками класса list, а так как неопределенные методы не были переопределены в своих классах, то будут работать так же, как и в классе list. Примером может служить метод len(), который вернет число элементов.

**Вывод.**

В ходе проделанной работы была построена система классов для градостроительной компании. Изучены элементы функционального и объектно-ориентированного программирования.

**Приложение А**
**Исходный код программ**

```
class HouseScheme:
    def __init__(self, count_rooms, area, bathroom_is):
        if type(count_rooms) != int or count_rooms < 0 or type(area) != int or area < 0
or type(bathroom_is) != bool:
            raise ValueError("Invalid value")
        self.count_rooms = count_rooms
        self.area = area
        self.bathroom_is = bathroom_is

class CountryHouse(HouseScheme):
    def __init__(self, count_rooms, area, bathroom_is, count_floors, area_site):
        super().__init__(count_rooms, area, bathroom_is)
        if type(count_floors) != int or count_floors < 0 or type(area_site) != int or
area_site < 0:
            raise ValueError("Invalid value")
        self.count_floors = count_floors
        self.area_site = area_site
    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.'.format(self.count_rooms, self.area, self.bathroom_is, self.count_floors,
self.area_site)
    def __eq__(self, other):
        if self.area == other.area and self.area_site == other.area_site and
abs(self.count_floors - other.count_floors) <= 1:
            return True
        else:
            return False

class Apartment(HouseScheme):
    def __init__(self, count_rooms, area, bathroom_is, count_floors, windows_where):
        super().__init__(count_rooms, area, bathroom_is)
        if type(count_floors) != int or count_floors < 1 or count_floors > 15 or
type(windows_where) != str or windows_where != 'N' and windows_where != 'S'
and windows_where != 'W' and windows_where != 'E':
            raise ValueError("Invalid value")
        self.count_floors = count_floors
        self.windows_where = windows_where
    def __str__(self):
```

```
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Этаж {}, Окна выходят на {}.'.format(self.count_rooms, self.area, self.bathroom_is, self.count_floors, self.windows_where)

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("Invalid type {}".format(type(p_object)))
    def total_square(self):
        return sum(list(map(lambda x: x.area, self)))

class ApartmentList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name
    def extend(self, iterable):
        for i in iterable:
            if isinstance(i , Apartment):
                super().append(i)
    def floor_view(self, floors, directions):
        for i in filter(lambda x: x.count_floors >= floors[0] and x.count_floors <= floors[1] and x.windows_where in directions, self):
            print("{}: {}".format(i.windows_where, i.count_floors))
```