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. 0382 _____ Афанасьев Н. С.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург
2020

Цель работы.

Изучение парадигм программирования, в особенности объектно-ориентированного, и исключений в языке Python.

Задание.

Построить систему классов для градостроительной компании, включающая классы *HouseScheme*, *CountryHouse*, *Apartment*, *CountryHouseList*, *ApartmentList*.

Выполнение работы.

Иерархия описанных классов приведена на рис. 1.

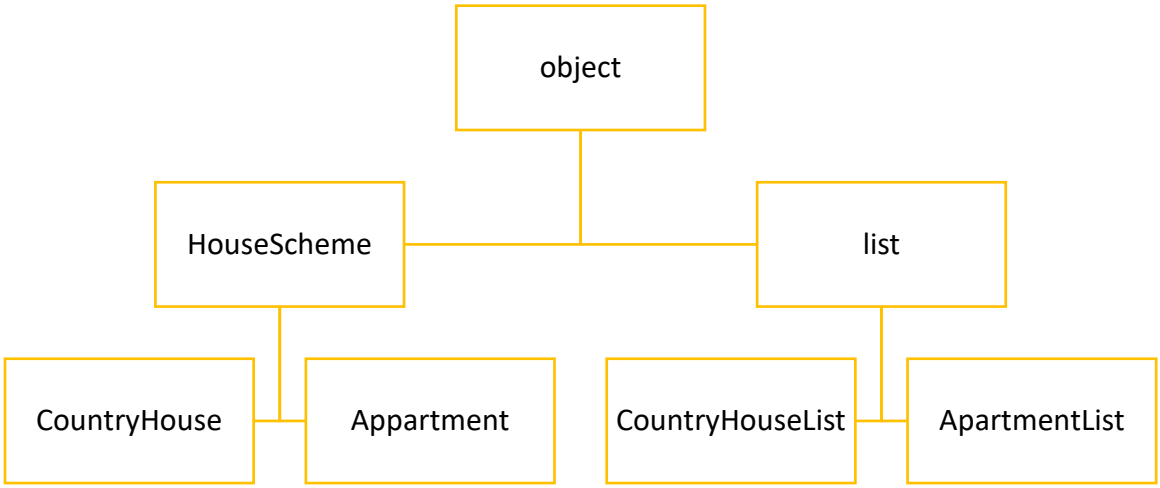

Рисунок 1 – Иерархия Классов

Класс **HouseScheme** имеет конструктор, принимающий основные параметры дома (*rooms*, *livingSpace*, *combinedBathroom*), проверяющий их валидность (иначе *ValueError*), записывающий значения в поля класса.

Дочерний от *HouseScheme* класс **CountryHouse** имеет схожую структуру, но ещё добавляются и проверяются значения *floors*, *area*. Переопределён метод *__str__* (метод *object*, вызывающийся при использовании *print()* или *str()*) так, чтобы выводились все параметры дома. Переопределён метод *__eq__* (метод *object*), который сравнивает дома по площадям и кол-ву этажей.

Дочерний от *HouseScheme* класс **Appartment** имеет схожую структуру, только принимаются значения *floor, side* и переопределён только метод *__str__*.

Дочерний от *list* класс **CountryHouseList** имеет конструктор, принимающий имя списка name, и вызывающий конструктор родительского класса. Переопределён метод append (метод *list*), который добавляет в список объект *CountryHouse* (иначе *TypeError*). Также есть метод *total_square*, возвращающий общую площадь домов.

Дочерний от *list* класс **AppartmentList** имеет такой же конструктор, как и прошлый класс. Переопределён метод extend (метод list), который добавляет объекты Appartment из входящего итерируемого объекта. Метод floor_view печатает те квартиры, которые удовлетворяют входящим данным.

Так как AppartmentList и CountryHouseList наследуются от list, будут работать и непереопределённые методы родительского класса. Пусть apt – объект класса AppartmentList. Тогда apt.reverse() перевернёт список, а apt.clear() очистит его.

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.	house1 = CountryHouse(6, 30, True, 2, 100) house2 = CountryHouse(8, 30, False, 3, 100) apt1 = Apartment(2, 20, True, 8, 'N') apt2 = Apartment(1, 10, False, 2, 'E') list1 = CountryHouseList('country') list1.append(house1) list1.append(house2) list2 = ApartmentList('apts') list2.extend([apt1, house1, apt2]) print(house1 == house2) print(apt2) print(list1.total_square()) list2.floor_view([5, 15], ['N', 'E'])	True Apartment: Количество жилых комнат 1, Жилая площадь 10, Совмещенный санузел False, Этаж 2, Окна выходят на E. 60 N: 8	Верно
2.	apt1 = Apartment('部屋', None , 'お手洗い', 'フロア', '北')	ValueError: Invalid value	Верно
3.	apt1 = Apartment(1, 10, False, 2, 'E') list1 = CountryHouseList('country') list1.append(apt1)	TypeError: Invalid type <class '__main__.Apartment'>	Верно

Выводы.

Были изучены парадигмы программирования и исключения в Python. Полученные знания были применены для решения поставленной задачи.

Разработана программа, которая содержит систему классов для градостроительной компании, включающая классы *HouseScheme*, *CountryHouse*, *Apartment*, *CountryHouseList*, *ApartmentLis*, описанные выше.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, rooms, livingSpace, combinedBathroom):
        if not (isinstance(rooms, int) and isinstance(livingSpace, int)
and livingSpace > 0 and isinstance(combinedBathroom, bool)):
            raise ValueError('Invalid value')
        else:
            self.rooms = rooms
            self.space = livingSpace
            self.comBath = combinedBathroom


class CountryHouse(HouseScheme):
    def __init__(self, rooms, livingSpace, combinedBathroom, floors,
area):
        if not (isinstance(floors, int) and isinstance(area, int)):
            raise ValueError('Invalid value')
        else:
            super().__init__(rooms, livingSpace, combinedBathroom)
            self.floors = floors
            self.area = area

    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая площадь
{}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.'.format(
            self.rooms,    self.space,    self.comBath,    self.floors,
self.area)

    def __eq__(self, other):
        if isinstance(other, CountryHouse):
            if self.space == other.space and self.area == other.area and
abs(self.floors - other.floors) <= 1: return True
        return False


class Apartment(HouseScheme):
    def __init__(self, rooms, livingSpace, combinedBathroom, floor,
side):
        if not (isinstance(floor, int) and (1 <= floor <= 15) and (side
in ['N', 'S', 'W', 'E'])):
            raise ValueError('Invalid value')
        else:
            super().__init__(rooms, livingSpace, combinedBathroom)
            self.floor = floor
            self.side = side

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {},
Совмещенный санузел {}, Этаж {}, Окна выходят на {}.'.format(
            self.rooms,    self.space,    self.comBath,    self.floor,
self.side)
```

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

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

    def total_square(self):
        return sum(house.space for house in self)


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

    def extend(self, iterable):
        for apt in iterable:
            if isinstance(apt, Apartment): super().append(apt)

    def floor_view(self, floors, directions):
        apts = list(filter(lambda apt: apt.floor in range(floors[0],
floors[1]+1) and apt.side in directions, self))
        for apt in apts: print(f"{apt.side}: {apt.floor}")
```