**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**](http://e.moevm.info/mod/quiz/view.php?id=119)

|  |  |  |
| --- | --- | --- |
| Студент гр. 0383 |  | Шквиря Е.В. |
| Преподаватель |  | Шевская Н.В. |

Санкт-Петербург

2020

## Цель работы.

Изучить устройство объектно-ориентированного подхода к программированию на языке Python и использование различных парадигм.

## 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().

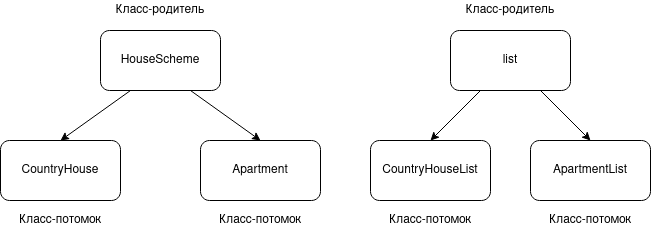
'''

## Выполнение работы.

Порядок выполнения поставленной задачи:

1. Для реализации классов, отведённых под здания, был создан класс-родитель (*HouseScheme*) с минимальным набором необходимых полей, от которого в последствии наследовались все дочерние классы (*CountryHouse, Apartament*) и дополнялись своими полями и переопределёнными методами. Конструкторы классов содержат проверку на корректность переданных аргументов и в случае некорректности данных могут выбросить исключение *ValueError*.
2. Для группировки зданий были реализованы классы *CountryHouseList* и *ApartamentList*, которые наследуются от класса *list*. В них были переопределены некоторые методы и добавлены свои.

Иерархия классов:



1. В ходе работы были перегружены методы:

Класс object:

\_\_init\_\_() - конструктор класса

\_\_str\_\_() - приведение класса к типу *str*

\_\_eq\_\_() - оператор «==»

Класс list:

append() - добавить объект в конец списка

extend() - добавить несколько объектов в конец списка

Используемые в программе переменные:

cnt\_rooms (cntRooms) — количество комнат

square\_room (squareRoom) — площадь комнаты

comb\_bath (combBath) — совмещённый санузел

cnt\_floors (cntFloors) — количество этажей

square\_land (squareLand) — площадь участка

floor — номер этажа

windows\_dir (windowsDir) — сторона, на которую выходит окно

name — название здания

p\_object — объект, который потенциально может быть добавлен в конец списка

sum — сумма площадей комнат

iterable — объект, из которого потенциально могут добавиться элементы в конец списка

floors — список с нижней и верхней границей допустимых этажей

directions — строка с допустимыми направлениями окон

low — нижняя граница допустимых этажей

up — верхняя граница допустимых этажей

it — объект-итератор, в котором хранятся отобранные элементы из ApartamentList

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. | HouseScheme(1,2,False) |  | Исключение остутствует |
| 2. | HouseScheme(3,4,“True“) |  | Произошло исключение |
| 3. | CountryHouse(1,2,True,4,5) |  | Исключение остутствует |
| 4. | CountryHouse(5,“4“,True,4,“5“) |  | Произошло исключение |
| 5. | Apartament(1,2,True,4,5) |  | Произошло исключение |
| 6. | Apartament(9,8,True,6,“S“) |  | Исключение остутствует |
| 7. | CountryHouseList(“abc“) |  | Исключение остутствует |
| 8. | ApartamentList(“abc“) |  | Исключение остутствует |

## Выводы.

Было изучено устройство объектно-ориентированного подхода к программированию на языке Python, методика использования наследования и полиморфизма. В основе использовалась императивная парадигма.

Разработана система классов для градостроительной компании, позволяющая задавать различные типы зданий и группировать их в списки. Для выполнения задачи использовался объектно-ориентированный подход, создание классов, переопределение методов и наследование.

**Доп. вопросы**.

3) Метод \_\_str\_\_() будет вызывать при попытке привести объект к типу str. Например, при попытке вывести этот элемент через функцию print()

4) Да, непереопределённые методы класса list будут работать для CountryHouseList и ApartamentList. В следствии того, что они не переопределены, они будут иметь реализацию, описанную в родительском классе, которая будет работать с полями, которые также описанны в родительском классе. Например, функция \_\_len\_\_() также будет считать количество элементов в списке, а \_\_eq\_\_() также будет проверять два списка на равенство объектов внутри них.

# Приложение А Исходный код программы

Название файла: lab3.py

class HouseScheme:  
 def \_\_init\_\_(self, cnt\_rooms, square\_room, comb\_bath):  
 if type(cnt\_rooms) is not int or type(square\_room) is not int or type(comb\_bath) is not bool or \  
 cnt\_rooms < 0 or square\_room < 0:  
 raise ValueError("Invalid value")  
 self.cntRooms = cnt\_rooms  
 self.squareRoom = square\_room  
 self.combBath = comb\_bath  
  
  
class CountryHouse(HouseScheme):  
 def \_\_init\_\_(self, cnt\_rooms, square\_room, comb\_bath, cnt\_floors, square\_land):  
 super().\_\_init\_\_(cnt\_rooms, square\_room, comb\_bath)  
 if type(cnt\_floors) is not int or type(square\_land) is not int or cnt\_floors < 0 or square\_land < 0:  
 raise ValueError("Invalid value")  
 self.cntFloors = cnt\_floors  
 self.squareLand = square\_land  
  
 def \_\_str\_\_(self):  
 return "Country House: Количество жилых комнат {}, Жилая площадь {}, " \  
 "Совмещенный санузел {}, Количество этажей {}, " \  
 "Площадь участка {}.".format(self.cntRooms, self.squareRoom, self.combBath, self.cntFloors,  
 self.squareLand)  
  
 def \_\_eq\_\_(self, other):  
 return self.squareRoom == other.squareRoom and self.squareLand == other.squareLand and abs(  
 self.cntFloors - other.cntFloors) <= 1  
  
  
class Apartment(HouseScheme):  
 def \_\_init\_\_(self, cnt\_rooms, square\_room, comb\_bath, floor, windows\_dir):  
 super().\_\_init\_\_(cnt\_rooms, square\_room, comb\_bath)  
 if type(floor) is not int or type(windows\_dir) is not str or floor < 1 or windows\_dir not in "NSWE" \  
 or floor > 15:  
 raise ValueError("Invalid value")  
 self.floor = floor  
 self.windowsDir = windows\_dir  
  
 def \_\_str\_\_(self):  
 return 'Apartment: Количество жилых комнат {}, Жилая площадь {}, ' \  
 'Совмещенный санузел {}, Этаж {}, Окна выходят на {}.' \  
 .format(self.cntRooms, self.squareRoom, self.combBath, self.floor, self.windowsDir)  
  
  
class CountryHouseList(list):  
 def \_\_init\_\_(self, name):  
 super().\_\_init\_\_()  
 self.name = name  
  
 def append(self, p\_object):  
 if type(p\_object) is not CountryHouse:  
 raise TypeError("Invalid type {}".format(type(p\_object)))  
 super().append(p\_object)  
  
 def total\_square(self):  
 sum = 0  
 for item in self:  
 sum += item.squareRoom  
 return sum  
  
  
class ApartmentList(list):  
 def \_\_init\_\_(self, name):  
 super().\_\_init\_\_()  
 self.name = name  
  
 def extend(self, iterable):  
 for item in iterable:  
 if type(item) is Apartment:  
 self.append(item)  
  
 def floor\_view(self, floors, directions):  
 low, up = floors  
 it = filter(lambda x: True if low <= x.floor <= up and x.windowsDir in directions else False, self)  
 for item in it:  
 print("{}: {}".format(item.windowsDir, item.floor))