**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. 9381 |  | Колованов Р.А. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2019

## Цель работы.

Изучение парадигм программирования, а также использование парадигмы ООП для написания программ.

## 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.

## Выполнение работы.

На схеме 1 представлена иерархия описанных классов:

Схема 1. Система классов для градостроительной компании

Класс *HouseScheme* имеет следующие поля:

* *room* – количество комнат.
* *area* – площадь.
* *bathroomAvailability* – наличие совмещенного санузла.

В конструкторе происходит проверка данных на корректность. Был переопределен метод \_\_init\_\_().

Класс *CountryHouse* имеет следующие поля:

* Поля, наследуемые из класса *HouseScheme*.
* *floors* – количество этажей.
* *landArea* – площадь участка.

В конструкторе происходит проверка данных на корректность. Были переопределены методы *\_\_str\_\_()*, *\_\_eq\_\_()* и *\_\_init\_\_()*..

Класс *Apartment* имеет следующие поля:

* Поля, наследуемые из класса *HouseScheme*.
* *floor* – этаж.
* *side* – сторона выхода окон.

В конструкторе происходит проверка данных на корректность. Были переопределены методы *\_\_str\_\_()* и *\_\_init\_\_()*.

В классе *CountryHouseList* переопределен метод *append()*, а также написан метод *total\_square()*.

В классе *ApartmenList* переопределен метод *extend()*, а также написан метод *floor\_view()*.

Метод *\_\_str\_\_()* вызывается при преобразовании к строке при помощи функции *str()*, при выводе на экран при помощи функции *print()*, а также при передаче объекта как аргумента функции *format()*. Если *\_\_str\_\_()* не определен, но *\_\_repr\_\_()* определен, то будет вызван метод *\_\_repr\_\_()*.

Не переопределенные методы класса *list* могут быть вызваны объектами производных классов. В данном случае, если мы попробуем вызвать у объекта класса *CountryHouseList* метод *count()*, то будет вызван метод *count()* класса *list*, так как мы унаследовали его. К примеру:

object = CountryHouseList()

print(object.count()) # Выведет 0

Разработанный программный код см. в приложении А.

## Тестирование.

Тестирование производилось на образовательной платформе Stepik.

## Выводы.

Были изучены основные парадигмы программирования. На языке Python, руководствуясь парадигмой ООП, была разработана программа, предоставляющая систему классов для градостроительной компании. Для написания программы использовались условные операторы, операторы цикла, классы, функция *filter()*, а также обработчик исключений *try-except*.

# Приложение А Исходный код программы

Название файла: main.py

class HouseScheme:

def \_\_init\_\_(self, rooms, area, bathroomAvailability):

if (area < 0) or ((bathroomAvailability is not True) and (bathroomAvailability is not False)):

raise ValueError("Invalid value")

self.rooms = rooms

self.area = area

self.bathroomAvailability = bathroomAvailability

class CountryHouse(HouseScheme):

def \_\_init\_\_(self, rooms, area, bathroomAvailability, floors, landArea):

HouseScheme.\_\_init\_\_(self, rooms, area, bathroomAvailability)

self.floors = floors

self.landArea = landArea

def \_\_str\_\_(self):

return "Country House: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка {}.".format(self.rooms, self.area, self.bathroomAvailability, self.floors, self.landArea)

def \_\_eq\_\_(self, other):

return (self.area == other.area) and (self.landArea == other.landArea) and (abs(self.floors - other.floors) <= 1)

class Apartment(HouseScheme):

def \_\_init\_\_(self, rooms, area, bathroomAvailability, floor, side):

HouseScheme.\_\_init\_\_(self, rooms, area, bathroomAvailability)

self.floor = floor

self.side = side

if self.floor < 1 or self.floor > 15:

raise ValueError("Invalid value")

if self.side not in ["N", "S", "W", "E"]:

raise ValueError("Invalid value")

def \_\_str\_\_(self):

return "Apartment: Количество жилых комнат {}, Жилая площадь {}, Совмещенный санузел {}, Этаж {}, Окна выходят на {}.".format(self.rooms, self.area, self.bathroomAvailability, self.floor, self.side)

class CountryHouseList(list):

def \_\_init\_\_(self, name):

list.\_\_init\_\_(self)

self.name = name

def append(self, p\_object):

if isinstance(p\_object, CountryHouse):

list.append(self, p\_object)

else:

raise TypeError("Invalid type {}".format(type(p\_object)))

def total\_square(self):

return sum([x.area for x in self])

class ApartmentList(list):

def \_\_init\_\_(self, name):

list.\_\_init\_\_(self)

self.name = name

def extend(self, iterable):

filteredIterable = filter(lambda x: isinstance(x, Apartment), iterable)

list.extend(self, filteredIterable)

def floor\_view(self, floors, directions):

filteredApartmentList = filter(lambda x: (x.floor >= floors[0]) and (x.floor <= floors[1]) and (x.side in directions), self)

for i in filteredApartmentList:

print("{}: {}".format(i.side, i.floor))