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

Студентка гр. 9304 _____ Каменская Е.К.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2019

Цель работы.

Изучить различные парадигмы программирования, в частности ООП. Реализовать свои структуры данных.

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.

Класс *HouseScheme*:

- Конструктор *__init__(self, rooms_count, indoor_area, bathroom_type)* принимает количество комнат, жилую площадь, тип санузла. Если количество комнат и жилая площадь неотрицательные и целочисленные, а тип санузла – логическая переменная, значения сохраняются в полях экземпляра класса, иначе выбрасывается исключение *ValueError("Invalid Value")*.

Класс *CountryHouse* (наследник *HouseScheme*):

- Конструктор *__init__(self, rooms_count, indoor_area, bathroom_type, floor, windows_dir)* принимает количество комнат, жилую площадь, тип санузла, которые передаются в конструктор суперкласса, число

этажей и площадь участка, которые сохраняются в полях экземпляра класса, если являются положительными целыми числами. Иначе выбрасывается исключение *ValueError(“Invalid Value”)*.

- Метод *__str__(self)* возвращает строковое представление полей объекта.
- Метод *__eq__(self, other)* получает на вход объект для сравнения с исходным и возвращает *True* или *False*, в зависимости от того, выполняются ли условия равенства двух объектов.

Класс *Apartment* (наследник *HouseScheme*):

- Конструктор *__init__(self, rooms_count, indoor_area, bathroom_type, floor, windows_dir)* получает на вход количество комнат, жилую площадь, тип санузла, – передаются конструктору суперкласса, – этаж и направление окон. Если этаж – целое число от 0 до 16 не включительно, а windows_dir имеет значение “N”, “S”, “W” или “E”, они сохраняются в полях экземпляра класса. Иначе выбрасывается исключение *ValueError(“Invalid Value”)*.
- Метод *__str__(self)* возвращает строковое представление полей объекта.

Класс *CountryHouseList* (наследник *list*):

- Конструктор __init__(self, name) получает на вход строку name, инициализирует созданный экземпляр как суперкласс и записывает строку name в поле name экземпляра класса.
- Метод *append(self, p_object)* получает на вход объект, который проверяет на принадлежность к классу *CountryHouse* и при выполнении этого условия добавляет объект в конец списка типа

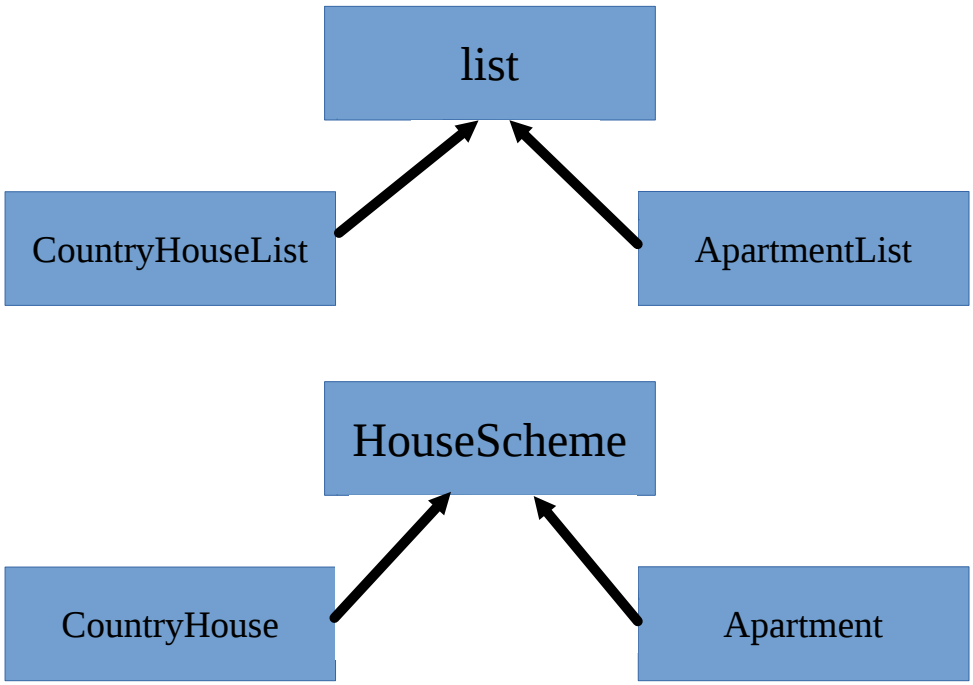
CountryHouseList, иначе выбрасывает исключение *TypeError(f"Invalid type {type(p_object)}")*.

- Метод *total_square(self)* суммирует значения поля *indoor_area* (жилая площадь) всех объектов списка и возвращает полученную сумму.

Класс *ApartmentList* (наследник *list*):

- Конструктор __init__(self, name) получает на вход строку name, инициализирует созданный экземпляр как суперкласс и записывает строку name в поле name экземпляра класса.
- Метод *extend(self, iterable)* проверяет каждый элемент iterable на принадлежность к классу *Apartment* и при выполнении этого условия добавляет объект в конец списка типа *ApartmentList*.
- Метод *floor_view(self, floors, directions)* с помощью функции *filter* и *lambda* функции отсеивает из списка те элементы, значения чьих полей *floor* и *windows_dir* соответствуют переданным в метод и выводит полученные результаты в виде <Направление>: <этаж>.

1. Иерархия классов



2. Переопределенные методы

class HouseScheme:

- __init__(self, rooms_count, indoor_area, bathroom_type):

class CountryHouse(HouseScheme):

- __init__(self, rooms_count, indoor_area, bathroom_type, floors_count, land_area)
- __str__(self)
- __eq__(self, other)

class Apartment(HouseScheme):

- __init__(self, rooms_count, indoor_area, bathroom_type, floor, windows_dir)
- __str__(self)

class CountryHouseList(list):

- __init__(self, name)
- append(self, p_object)
- total_square(self)

class ApartmentList(list):

- __init__(self, name)
- extend(self, iterable)
- floor_view(self, floors, directions)

3. В каких случаях будет вызван метод __str__().

Метод __str__() вызывается, когда необходимо получить строковое представление объекта, вывести его в виде строки.

4. Непереопределенные методы

Непереопределенные методы класса list будут работать для объектов CountryHouseList и ApartmentList так, как они работают для объектов базового класса list. Например, для объектов обоих подклассов будут работать методы reverse() и pop().

Разработанный программный код см. в приложении А.

Выводы.

В ходе выполнения работы были изучены различные парадигмы программирования. Была написана программа с использованием ООП, реализовано наследование классов. Ошибочный ввод данных отлавливается искусственно выбрасываемыми исключениями.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: LAB3.py

```
class HouseScheme:
    def __init__(self, rooms_count, indoor_area, bathroom_type):
        if (type(rooms_count) == int) and (rooms_count >= 0) and \
           (type(indoor_area) == int) and (indoor_area >= 0) and \
           (type(bathroom_type) == bool):
            self.rooms_count = rooms_count
            self.indoor_area = indoor_area
            self.bathroom_type = bathroom_type
        else:
            raise ValueError("Invalid value")

class CountryHouse(HouseScheme):
      def __init__(self, rooms_count, indoor_area, bathroom_type,
floors_count, land_area):
        super().__init__(rooms_count, indoor_area, bathroom_type)
        if (type(floors_count) == int) and \
           (floors_count > 0) and \
           (type(land_area) == int) and \
           (land_area >= indoor_area):
            self.floors_count = floors_count
            self.land_area = land_area
        else:
            raise ValueError("Invalid value")

    def __str__(self):
            return "Country House: Количество жилых комнат " +
str(self.rooms_count) + \
                ", Жилая площадь " + str(self.indoor_area) + \
                ", Совмещенный санузел " + str(self.bathroom_type) +
\
                ", Количество этажей " + str(self.floors_count) + \
                ", Площадь участка " + str(self.land_area) + "."

    def __eq__(self, other):
      return (self.indoor_area == other.indoor_area) and \
                (self.land_area == other.land_area) and \
                (abs(self.floors_count - other.floors_count) <= 1)

class Apartment(HouseScheme):
      def __init__(self, rooms_count, indoor_area, bathroom_type,
floor, windows_dir):
        super().__init__(rooms_count, indoor_area, bathroom_type)
            if (type(floor) == int) and (0 < floor < 16) and
(windows_dir in ['N','S','W','E']):
            self.floor = floor
            self.windows_dir = windows_dir
        else:
            raise ValueError("Invalid value")

    def __str__(self):
            return "Apartment: Количество жилых комнат " +
str(self.rooms_count) + \
```

```
                    ", Жилая площадь " + str(self.indoor_area) + \
                     ", Совмещенный санузел " + str(self.bathroom_type) +
\
                    ", Этаж " + str(self.floor) + \
                    ", Окна выходят на " + str(self.windows_dir) + "."

    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(f"Invalid type {type(p_object)}")

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

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

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

        def floor_view(self, floors, directions):
              for flat in list(filter(lambda x: (x.floor >= floors[0])
and \
                                                (x.floor <= floors[1])
and \
                                                     (x.windows_dir in
directions), self)):
                print(f"{flat.windows_dir}: {flat.floor}")
```