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 _____ Крючков А.М.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Создать систему классов для градостроительной компании, используя парадигму ООП на языке Python.

Задание.

Создать систему классов для градостроительной компании:

HouseScheme - базовый класс (схема дома);

- Поля класса:
- количество жилых комнат
- жилая площадь (в квадратных метрах)
- совмещенный санузел (значениями могут быть или False, или True)
- При несоответствии переданного значения вызвать исключение.

CountryHouse — деревенский дом (наследник HouseScheme);

Поля класса:

- все поля класса HouseScheme
- количество этажей
- площадь участка

При несоответствии переданного значения вызвать исключение.

Переопределяемые методы:

- __str__()
- __eq__()

Apartment — квартира городская (наследник HouseScheme);

Поля класса:

- все поля класса HouseScheme
- этаж (может быть число от 1 до 15)
- куда выходят окна (значением может быть одна из строк: N, S, W, E))

  При несоответствии переданного значения вызвать исключение.

Определяемые методы:

- __str__()

CountryHouseList — список деревенских домов (наследник list);

Конструктор:

Вызвать конструктор базового класса

a. Передать в конструктор строку name и присвоить её полю name созданного объекта

- 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>

- total_square():

  Посчитать общую жилую площадь.

1. ApartmentList — список городских квартир (наследник list);

- Методы:

  Конструктор:

  a. Вызвать конструктор базового класса
  b. Передать в конструктор строку name и присвоить её полю name
  c. созданного объекта

- extend(iterable):

  a. Переопределение метода extend() списка. В случае, если элемент
  b. iterable - объект класса Apartment, этот элемент добавляется в
  c. список, иначе не добавляется.

- floor_view(floors, directions):

  В качестве параметров метод получает диапазон возможных этажей в виде списка. Метод должен выводить квартиры, этаж которых входит в диапазон и окна которых выходят в одном из направлений.

Основные теоретические положения.

Объектно-ориентированная парадигма базируется на нескольких принципах: наследование, инкапсуляция, полиморфизм. Наследование - специальный механизм, при котором мы можем расширять классы, усложняя их функциональность. В наследовании могут участвовать минимум два класса: суперкласс (или класс-родитель, или базовый класс) - это такой класс, который

был расширен. Все расширения, дополнения и усложнения класса-родителя реализованы в классе-наследнике (или производном классе, или классе потомке) - это второй участник механизма наследования. Наследовани епозволяет повторно использовать функциональность базового класса, при этомне меняя базовый класс, а также расширять ее, добавляя новые атрибуты.

Выполнение работы.

Были созданы следующие классы.

1. HouseScheme

   Поля:

   - self. number_of_living_rooms — количество комнат;
   - self.area — жилая площадь;
   - self.combwc— совмещенный санузел;

2. CountryHouse(наследник HouseScheme)

   Поля:

   - self. number_of_floors — количество этажей;
   - self.land_area — площадь земельного участка;

   Методы:

   - __str__ - возвращает отформатированную строку с информацией об объекте.
   - __eq__ - метод сравнения двух классов.

3. Apartment(наследник HouseScheme)

   Поля:

   - self.floor — этаж, на котором располагается квартира;
   - self.side— направление, куда выходят окна квартиры;

   Методы:

   - __str__ - возвращает отформатированную строку с информацией об объекте.

4. CountryHouseList(наследник list)

   Поля:

   - self.name – имя списка.

Методы:

- append- добавляет элемент в список.
- total_square – возвращает, сумму площади домов всех CountryHouse в списке.

5. ApartmentList(наследник list)

Поля:

- self.name – имя списка.

Методы:

- extend - применяет метод extend родителя, только для элементов класса Apartment.
- floor_view – позволяет узнать, выходят ли окна квартиры (объекта класса Apartment) на одну из требуемых сторон и находится ли квартира на одном из требуемых этажей.

1. Иерархия классов

   Родитель: HouseScheme

   - Наследники: Apartment, CountryHouse,

   Родитель: list

   - Наследники: ApartmentList, CountryHouseList.

2. Методы базовых классов, которые переопределялись при выполнении:
   - __str__(self,..);
   - __eq__(self, other) (бинарный оператор «==»);
   - append(self, p_object);
   - extend(self, lst);
   - __init__(self,…);
3. Метод __str__() будет вызван при приведении объекта класса к строковому типу (например при вызове функции str() или при использовании в функции print()).

Не переопределённые методы класса list для классов CountryHouseList и ApartmentList работать будут, потому что ApartmentList и CountryHouseList являются наследниками класса list, поэтому все методы, присущие классу list будут также работать для всех объектов классов-наследников. Например не переопределённый метод self.reverse() развернет список.

Выводы.

В ходе работы было выполнено задание — построена система классов для градостроительной компании.

Реализована система классов, представлена иерархия классов, переопределены все необходимые методы классов, инициализированы поля объектов классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, number_of_living_rooms, area, combwc):
        if area < 0 or type(combwc)!=bool:
            raise ValueError("Invalid value")
        self.number_of_living_rooms = number_of_living_rooms
        self.area = area
        self.combwc = combwc
class CountryHouse(HouseScheme):
    def __init__(self, number_of_living_rooms, area, combwc,
number_of_floors, land_area):
        super().__init__(number_of_living_rooms, area, combwc)
        self.number_of_floors = number_of_floors
        self.land_area = land_area
    def __str__(self):
        return "Country House: Количество жилых комнат {}, Жилая площ
адь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.".format(self.number_of_living_rooms, self.area, self.combwc,
self.number_of_floors, self.land_area)
    def __eq__(self, another_house):
        return self.area==another_house.area and self.land_area ==
another_house.land_area and abs(self.number_of_floors-
another_house.number_of_floors)<=1
class Apartment(HouseScheme):
    def __init__(self, number_of_living_rooms, area, combwc, floor,
side):
        super().__init__(number_of_living_rooms, area, combwc)
        if 1<= floor <= 15 and side in ['N', 'S', 'W', 'E']:
            self.floor = floor
            self.side = side
        else:
            raise ValueError('Invalid value')
    def __str__(self):
        return "Apartment: Количество жилых комнат {}, Жилая площадь
{}, Совмещенный санузел {}, Этаж {}, Окна выходят на
{}.".format(self.number_of_living_rooms, self.area, self.combwc,
self.floor, self.side)
class CountryHouseList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name
    def append(self, p_object):
        if CountryHouse != type(p_object):
            raise TypeError("Invalid type {}".format(type(p_object)))
        super().append(p_object)
    def total_square(self):
        res = 0
        for house in self:
            res+=house.area
        return res
class ApartmentList(list):
```

```
        def __init__(self, name):
            super().__init__()
            self.name = name
        def extend(self, iterable):
            super().extend(list(filter(lambda   item:   type(item)   ==
Apartment, iterable)))
        def floor_view(self, floors, directions):
            for side_floor in list(filter(lambda apart: floors[0] <=
apart.floor <= floors[1]  and apart.side in directions, self)):
                print('{}: {}'.format(side_floor.side, side_floor.floor))
```