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 _____ Литягин С.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:

- Базовый класс - схема дома HouseScheme:
  - Поля объекта класса HouseScheme:
    1. количество жилых комнат
    2. площадь (в квадратных метрах, не может быть отрицательной)
    3. 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 (наследник HouseScheme):
  - Поля объекта класса CountryHouse:
    1. количество жилых комнат
    2. жилая площадь (в квадратных метрах)
    3. совмещенный санузел (значениями могут быть или False, или True)
    4. количество этажей
    5. площадь участка

При создании экземпляра класса CountryHouse необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

  - Переопределение методов __str__() и __eq__()
- Квартира городская Apartment (наследник HouseScheme):
  - Поля объекта класса Apartment:
    1. количество жилых комнат
    2. площадь (в квадратных метрах)
    3. совмещенный санузел (значениями могут быть или False, или True)
    4. этаж (может быть число от 1 до 15)

5. 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'.

- Переопределение метода __str__()

- CountryHouseList - список деревенских домов (наследуется от класса list)
  - Конструктор:
    1. Вызвать конструктор базового класса
    2. Передать в конструктор строку name и присвоить её полю name созданного объекта
  - Переопределение метода append(p_object) списка. В случае, если p_object - деревенский дом, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>'".
  - Определение метода total_square(), чтобы посчитать общую жилую площадь
- 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
  - Переопределение метода extend(iterable) списка. В случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.
  - Определение метода floor_view(floors, directions). Метод должен выводить квартиры, этаж которых входит в переданный диапазон и окна которых выходят в одном из переданных направлений.

Основные теоретические положения.

Объектно-ориентированная парадигма базируется на нескольких принципах: наследование, инкапсуляция, полиморфизм. Наследование – специальный механизм, при котором мы поможем 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. Наследование позволяет повторно использовать функциональность базового класса, при этом не меняя базовый класс, а также расширять ее, добавляя новые атрибуты.

Основные понятия ООП:

- Объект – конкретная сущность предметной области
- Класс – тип объекта (объект - экземпляр класса)
- Метод класса – функция, которая принадлежит классу (*<имя_объекта>.<метод>(аргумент_1, аргумент_2,..)*)
- Конструктор – метод, который вызывается при создании экземпляра класса. Синтаксис создания: *def __init__(self, <arg_1>, <arg_2>)*
- Поле объекта – некоторая переменная, которая лежит в области видимости объекта и доступна во внешней программе через синтаксис: *<имя_объекта>.<поле>*

Функция *filter(<функция>, <объект>)* – возвращает объект-итератор, состоящий из тех элементов итерируемого объекта <объект>, для которых <функция> является истиной.

Лямбда-выражение – особый синтаксис в Python, позволяющая упростить запись и использование однострочных операций.

Команда *raise*. Генерирует исключение (бросает исключение), которое должно быть объектом класса Exception.

Выполнение работы.

Создается главный класс-родитель *HouseScheme*. Затем определяется конструктор с учетом аргументов. Если аргументы соответствуют определенным требованиям, то в конструкторе инициализируются поля объекта класса:

- *self.rooms* – количество комнат
- *self.square* – жилая площадь
- *self.bathroom* – совмещенный санузел

В противном случае оператором raise вызывается исключение *ValueError* с текстом “Invalid value”.

Создается класс *CountryHouse* (наследуется от *HouseScheme*). Помимо полей класса-родителя, в этом классе в конструкторе (если аргументы соответствуют определенным условием, иначе вызывается исключение *ValueError* с текстом “Invalid value”) инициализируются следующие поля объекта класса:

- *self.floors* – количество этажей
- *self.square_place* – площадь земельного участка

Также переопределяется метод *__str__()*, который должен возвращать отформатированную строку с информацией об объекте, и метод *__eq__()*.

Создается класс *Apartment* (наследуется от *HouseScheme*). Помимо полей класса-родителя, в этом классе в конструкторе (если аргументы соответствуют определенным условием, иначе вызывается исключение *ValueError* с текстом “Invalid value”) инициализируются следующие поля объекта класса:

- *self.floors* – количество этажей
- *self.window* – направление, куда выходят окна квартиры

Также переопределяется метод *__str__()*, который должен возвращать отформатированную строку с информацией об объекте.

Создается класс *CountryHouseList* (наследник *list*). В конструкторе этого класса определяется поле *self.name*, а также вызывается конструктор класса-родителя. Также переопределяется метод *append()*. Если объект имеет тип

CountryHouse, то происходит добавление, иначе вызывается ошибка. А еще определяется метод *total_square()*, в котором подсчитывается общая жилая площадь.

Создается класс *ApartmentList* (наследуется от *list*). В конструкторе этого класса определяется поле *self.name*, а также вызывается конструктор класса-родителя. Также переопределяется метод *extend()* в соответствии с условиями. А еще определяется метод *floors_view()*, который позволяет узнать, выходят ли окна квартире на одну из требуемых сторон, и находится ли квартира на одном из требуемых этажей.

Иерархия классов:

- Класс-родитель *HouseScheme*:
  Классы-наследники: *CountryHouse*, *Apartment*
- Класс-родитель *list*:
  Классы-наследники: *CountryHouseList*, *ApartmentList*
  Переопределенные методы:
- *__init__(self, ...)*
- *__str__(self, ...)*
- *__eq__(self, other)*
- *append(self, p_object)*
- *extend(self, iterable)*

Метод *__str__()* будет вызван, если объект приводят к строковому типу (при вызове *str()* или использовании *print()*).

Непереопределенные методы класса *list* для *CountryHouseList* и *ApartmentList* будут работать, поскольку они являются наследниками *list*. Например, метод *self.reverse()* будет работать с объектами вышеперечисленных классов, а именно – будет разворачивать список.

Тестирование.

Для тестирования в программу добавлен следующий код:

```
house1 = CountryHouse(4, 65.3, False, 2, 103)
house2 = CountryHouse(4, 45, True, 3, 120)
house_list = CountryHouseList("Седьмое небо")

apart1 = Apartment(3, 35, True, 7, 'N')
print(apart1)

print(house1)
house_list.append(house1)
house_list.append(house2)
print(house_list.total_square())
```

Результаты тестирования представлены в таблице 1:

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	…	Apartment: Количество жилых комнат 3, Жилая площадь 35, Совмещенный санузел True, Этаж 7, Окна выходят на N. Country House: Количество жилых комнат 4, Жилая площадь 65.3, Совмещенный санузел False, Количество этажей 2, Площадь участка 103. 110.3	Программа работает правильно

Выводы.

В ходе работы была создана система классов для градостроительной компании с использованием парадигмы ООП на языке Python.

Реализована система классов, инициализированы поля объектов классов, переопределены все необходимые методы классов, представлена иерархия классов, инициализированы поля объектов классов. Помимо этого были предусмотрены возникновения исключительных ситуаций и реализован вывод соответствующих исключений. Были использованы парадигмы функционального программирования на Python (lambda-выражения и функция filter()).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, rooms, square, bathroom):
        if square > 0 and rooms > 0 and type(rooms) == int and
type(bathroom) == bool:
            self.rooms = rooms
            self.square = square
            self.bathroom = bathroom
        else:
            raise ValueError("Invalid value")

class CountryHouse(HouseScheme):
    def __init__(self, rooms, square, bathroom, floors,
square_place):
        super().__init__(rooms, square, bathroom)
        if floors > 0 and type(floors) == int and square_place > 0:
            self.floors = floors
            self.square_place = square_place
        else:
            raise ValueError("Invalid value")

    def __str__(self):
        return ("Country House: Количество жилых комнат {}, Жилая
площадь {}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.".format(self.rooms, self.square, self.bathroom, self.floors,
self.square_place))

    def __eq__(self,others):
        return self.square == others.square and self.square_place
== others.square_place and abs(self.floors - others.floors) <= 1

class Apartment(HouseScheme):
    def __init__(self, rooms, square, bathroom, floors, window):
        windows = ['N', 'S', 'W', 'E']
        super().__init__(rooms, square, bathroom)
        if 15>= floors > 0 and type(floors) == int and (window in
windows):
            self.floors = floors
            self.window = window
        else:
            raise ValueError("Invalid value")

    def __str__(self):
        return ("Apartment: Количество жилых комнат {}, Жилая
площадь {}, Совмещенный санузел {}, Этаж {}, Окна выходят на
{}.".format(self.rooms, self.square, self.bathroom, self.floors,
self.window))

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("Invalid type {}".format(type(p_object)))

        def total_square(self):
            square_total = 0
            for i in self:
                square_total += i.square
            return square_total
    class ApartmentList(list):
        def __init__(self, name):
            super().__init__()
            self.name = name

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

        def floor_view(self, floors, directions):
            for i in filter(lambda x: floors[0] <= x.floors <= floors[1]
and x.window in directions, self):
                print("{}: {}".format(i.window, i.floors))
```