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 _____ Куликов М.Д.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

Цель работы.

Получение опыта создания систем классов с использованием исключений и некоторых составляющих функционального программирования.

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

Класс должен наследоваться от 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'

Переопределить методы __str__ и __eq__

Квартира городская Apartment:

Класс должен наследоваться от HouseScheme

Поля объекта класса Apartment:

1)количество жилых комнат

2)площадь (в квадратных метрах)

4)совмещенный санузел (значениями могут быть или False, или True)

5)этаж (может быть число от 1 до 15)

6)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'

Переопределить метод __str__

Деревня:

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

Определить метод total_square (считает общую жилую площадь)

Переопределитть метод append:

В случае, если p_object - деревенский дом, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>

Жилой комплекс:

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

Определить метод floor_view:

Метод должен выводить квартиры, этаж которых входит в переданный диапазон и окна которых выходят в одном из переданных направлений.

Переопределить метод extend:

В случае, если элемент iterable - объект класса Apartment, этот элемент добавляется в список, иначе не добавляется.

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. Наследование позволяет повторно использовать функциональность базового

класса, при этом не меняя базовый класс, а также расширять ее, добавляя новые атрибуты.

Класс — тип объекта

Метод класса — функция, определяемая внутри класса

Объект класса - конкретная сущность предметной области

Функция filter() применяет функцию ко всем элементам последовательности и возвращает итератор с теми объектами, для которых функция вернула True.

Лябмда-выражения — это особый синтаксис в Python, необходимый для создания анонимных функций. Лямбда-выражения в Python позволяют функции быть созданной и переданной (зачастую другой функции) в одной строчке кода.

raise - Инструкция позволяет прервать штатный поток исполнения при помощи возбуждения исключения.

Выполнение работы.

Был создан главный класс-родитель HouseScheme, от когорого будут наследоваться классы CountryHouse и Apartment. Далее был определен конструктор с учетом принимаемых аргументов. Аргументы были проверены на соответствие типов и значений и записаны в соответствующие поля обхекта класса. При несоответствии типов или значений будет выброшено исключение ValueError с сообщением «Invalid value»

Был создан класс CounrtyHouse , наследующийся от класса HouseScheme. В нем был определен конструктор с использованием конструктора класса — родителя. Соответствующие аргументы записываются в соответствующие поля. Далее был переопределен метод __str__ и __eq__ в соответствии с условием задания.

Был создан класс Apartment, наследующийся от класса HouseScheme.

Было определено поле класса с возможными направлениями выхода окон. Был определен конструктор с использованием конструктора класса — родителя. Аргументы были проверены критериями из условия и записаны в соответствующие поля. При несоответствии типов или значений будет выброшено исключение ValueError с сообщением «Invalid value». Был переопределен метод __str__ в соответствии с условием.

Был создан класс CountryHouseList, наследующийся от стандартного класса list. Был определен конструктор с использованием конструктора класса — родителя. Аргументы были записаны в соответствующие поля. Был переопределен метод append в соответствии с условием и был определен метод total_area, вычисляющий общую жилую площадь.

Был создан класс ApartmentList, наследующийся от стандартного класса list. В нем был определен конструктор с использованием конструктора класса — родителя. Был переопределен метод extend в соответствии с условиями и определен метод floor_view, который выводит этажи и направления окон квартир, соответствующим условиям аргументов.

Иерархия классов:

Классы Apartment и CountryHouseнаследуются от класса HouseScheme.

Классы CountryHouseList и ApartmentList наследуются от стандартного класса list.

Переопределенные методы можно увидеть выше, в описании классов.

Метод __str__ будет вызван при использовании функции str или при распечатывании объекта с помощью print.

Непереопределенные методы для этих классов будут работать, так как они являются наследниками класса list и у них будут работать все методы,

определенные для него. К примеру, если использовать метод clear, то объекты классов ApartmentList и CountryHouseList станут пустыми.

Тестирование.

Для проведения тестирования в программу был добавлен следующий код:

```
country = CountryHouse(5, 64, True, 2, 120)
country1 = CountryHouse(4, 54, False, 3, 120)
apart = Apartment(3, 50, False, 10, 'N')
apart1 = Apartment(3, 25, False, 4, 'S')
country_list = CountryHouseList('country123')
apart_list = ApartmentList('apart123')

country_list.append(country)
country_list.append(country1)
apart_list.append(apart)
apart_list.append(apart1)
apart_list.extend(country_list)

print(country_list.total_square())
print(country)
print(apart)
print(country_list)
print(apart_list)
```

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	---	118 Country House: Количество жилых комнат 5, Жилая площадь 64, Совмещенный санузел True, Количество этажей 2, Площадь участка 120. Apartment: Количество жилых комнат 3, Жилая площадь 50, Совмещенный санузел False, Этаж 10, Окна выходят на N. [<__main__.CountryHouse object at 0x7fdf77b07670>, <__main__.CountryHouse object at 0x7fdf77b076a0>] [<__main__.Apartment object at 0x7fdf77b07760>, <__main__.Apartment object at 0x7fdf77b077c0>]	Корректная работа программы.

Выводы.

Был получен опыт создания систем классов с использованием исключений и некоторых составляющих функционального программирования.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class HouseScheme:
    def __init__(self, room_amount, area, comb_bath):
        if type(comb_bath) == bool and area >= 0:
            self.room_amount = room_amount
            self.area = area
            self.comb_bath = comb_bath
        else:
            raise ValueError('Invalid value')



class CountryHouse(HouseScheme):
    def __init__(self, room_amount, area, comb_bath, floors_amount,
land_area):
    super().__init__(room_amount, area, comb_bath)
    self.floors_amount = floors_amount
    self.land_area = land_area



    def __str__(self):
        return "Country House: Количество жилых комнат {}, Жилая
    площадь {}, Совмещенный санузел {}, Количество " \
        "этажей {}, Площадь участка {}.".format(self.room_amount,
    self.area, self.comb_bath, self.floors_amount, self.land_area)


    def __eq__(self, other):
        return self.area == other.area and self.land_area ==
other.land_area and abs(self.floors_amount - other.floors_amount) <= 1
```

```
class Apartment(HouseScheme):
    window_directions = ['N', 'S', 'W', 'E']

    def __init__(self, room_amount, area, comb_bath, floor, win_dir):
        super().__init__(room_amount, area, comb_bath)
        if (1 <= floor <= 15) and (win_dir in Apartment.window_directions):
            self.floor = floor
            self.win_dir = win_dir
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь
         {}, Совмещенный санузел {}, Этаж {}, ' \
              'Окна выходят на {}.'.format(self.room_amount, self.area,
         self.comb_bath, self.floor, self.win_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("Invalid type {}".format(type(p_object)))

    def total_square(self):
```

```
        total_area = 0
        for i in self:
            total_area += i.area
        return total_area



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.floor <= floors[1] and x.win_dir in
directions, self):
            print('{}: {}'.format(i.win_dir, i.floor))
```