МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Информатика»

ТЕМА: ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2020

Цель работы.

Изучить основные принципы работы Машины Тьюринга, а также отработать ее реализацию и моделирование на языке Python.

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(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(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(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(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().

Основные теоретические положения.

ООП основывается на наследовании, инкапсуляции и полиморфизме. Используя наследование, 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.

Синтаксис функции: filter(<функция>, <объект>) Функция применяется для каждого элемента итерируемого объекта <объект> и возвращает объект-итератор, состоящий из техэлементов итерируемого объекта <объект>, для которых <функция>является истиной.

Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций. Синтаксис определения следующий: lambda аргумент1, аргумент2,..., аргументN : выражение

Выполнение работы.

Класс *HouseScheme().*Классы-потомки *Apartment* и *CountryHouse*. В конструкторе инициализируются поля объекта класса *quant_rooms*(количество жилых комнат), *area_in*(жилая площадь), *san_uz*(совмещенный санузел (значениями могут быть или *False*, или *True)*). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Класс *CountryHouse(HouseScheme)*. В конструкторе наследуются поля объекта класса *HouseScheme* - *quant_rooms , area_in, san_uz* и инициализируются другие поля – *quant_floors*(количество этажей), *area_out(*площадь участка). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Переопределим метод *__str__(self)*, который возвращает строку заданного формата, и метод *__eq__(self, other)*, который возвращает *True*, если два объекта класса, переданные в метод, равны, и *False* в противном случае.

Класс *Apartment(HouseScheme)*. В конструкторе наследуются поля объекта класса *HouseScheme* - qua*nt_rooms , area_in, san_uz* и инициализируются другие поля – *floor*(этаж (может быть число от 1 до 15)), *windows*(куда выходят окна (значением может быть одна из строк: N, S, W, E)). Осуществляется проверка переданных в конструктор параметров. Если они не удовлетворяют требованиям, с помощью *raise* создаётся и выбрасывается исключение *ValueError* с текстом 'Invalid value' .

Затем переопределяется метод *__str__(self)*, который возвращает строку заданного формата.

Класс *CountryHouseList(list)*. В конструкторе инициализируется поле объекта класса – *name* (полю класса присваивается аргумент-строки *name*). Затем происходит переопределение метода *append(self, p_object)*, в котором осуществляется проверка переданного аргумента. Если переданный в метод аргумент *p_object* удовлетворяет заданным условиям, то элемент добавляется в список, иначе выбрасывается исключение *TypeError* с текстом: *'Invalid type <тип_объекта p_object>'*. Также происходит переопределение метод *total_square(self)*, в котором считается и возвращается общая жилая площадь текущего объекта класса.

Класс *ApartmentList(list)*. Потомок класса *list*, не является родителем. В конструкторе инициализируется поле объекта класса – *name*(полю класса присваивается аргумент-строки *name*). Затем происходит переопределение метода списка - *extend(self, iterable)*, в котором осуществляется проверка, элемента *iterable.* Если элемент *iterable* - объект класса *Apartment,* он добавляется в список. Также переопределяется метод *floor_view(self, floors, directions)*. Метод в качестве параметров получает диапазон возможных этажей в виде списка и список направлений из ('N', 'S', 'W', 'E'). Метод выводит квартиры, удовлетворяющие заданным условиям, окна которых выходят в одном из переданных направлений, преобразуя их в строку заданного формата.

1. Иерархия описанных классов.

CountryHouse(потомок) – *HouseScheme*(родитель)

*Apartmen*t(потомок) – *HouseScheme*(родитель)

CountryHouseList(потомок) - *list*(родитель)

ApartmentList(потомок) - *list*(родитель)

2. Методы, которые были переопределены:

def __init__(self,);

def __str__(self);

def __eq__(self, other);

def append(self, p_object);

def extend(self, iterable).

3. Метод *__str__*() будет вызван:

При вызове функции *str*() - приведении к типу “строка” в явном виде, или неявном, как, например, при вызове функции *print()*.

4. Будут ли работать непереопределенные методы класса *list* для *CountryHouseList* и *ApartmentList*? Объясните почему и приведите примеры.

Непереопределенные методы также будут работать, но в их базовом формате, как обычные функции класса *list,* так как он является родителем классов *CountryHouseList* и *ApartmentList.*

Пример: метод *list.pop*(), если его не переопределить, будет удалять i-ый элемент и возвращать его. Если индекс не указан, удалится последний элемент списка *CountryHouseList* или *ApartmentList.*

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	print(house1)	Country House: Количество жилых комнат 5, Жилая площадь 55, Совмещенный санузел True, Количество этажей 2, Площадь участка 900.	Вывод верный
2.	print(apartment1)	Apartment: Количество жилых комнат 3, Жилая площадь 70, Совмещенный санузел False, Этаж 9, Окна выходят на S.	Вывод верный

Выводы.

Были освоены парадигмы программирования, а также отработаны на практике их реализации в программном коде на языке Python.

Разработана программа, описывающая классы и переопределяющая некоторые методы. Также программа отлавливает возможные исключения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

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

class CountryHouse(HouseScheme):
    def
__init__(self,quant_rooms,area_in,san_uz,quant_floors,area_out):
        super().__init__(quant_rooms,area_in,san_uz)
        if isinstance(quant_floors,int) and isinstance(area_out,int):
            self.quant_floors=quant_floors
            self.area_out=area_out
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return 'Country House: Количество жилых комнат {}, Жилая площадь
{}, Совмещенный санузел {}, Количество этажей {}, Площадь участка
{}.'.format(self.quant_rooms,        self.area_in,        self.san_uz,
self.quant_floors, self.area_out)

    def __eq__(self,other):
       if self.area_in==other.area_in and self.area_out==other.area_out
and abs(self.quant_floors-other.quant_floors)<=1:
           return True
       else:
           return False


class Apartment(HouseScheme):

    def __init__(self,quant_rooms,area_in,san_uz,floor,windows):
        super().__init__(quant_rooms,area_in,san_uz)
        if isinstance(floor,int) and floor<16 and floor>0 and windows
in ['N','S','W','E']:
            self.floor=floor
            self.windows=windows
        else:
            raise ValueError('Invalid value')

    def __str__(self):
        return 'Apartment: Количество жилых комнат {}, Жилая площадь {},
Совмещенный    санузел    {},    Этаж    {},    Окна    выходят    на
{}.'.format(self.quant_rooms,  self.area_in,  self.san_uz,  self.floor,
self.windows)


class CountryHouseList(list):
```

```
    def __init__(self,name):
        super().__init__()
        self.name=name

    def append(self,p_object):
        if issubclass(type(p_object),CountryHouse):
            super().append(p_object)
        else:
            st='Invalid type '+str(type(p_object))
            raise TypeError(st)

    def total_square(self):
        total_sq=0
        for i in self:
            total_sq+=i.area_in
        return total_sq


class ApartmentList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self,iterable):
        super().extend(filter(lambda apart: type(apart) == Apartment,
iterable))

    def floor_view(self,floors, directions):
        filtered=list(filter(lambda     apart:     (apart.windows     in
directions)              and              (apart.floor              in
list(range(floors[0],floors[1]+1))),self))
        for i in filtered:
            print("{}: {}".format(i.windows,i.floor))
```