**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. 0382 _____ Чегодаева Е.А.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2020

**Цель работы.**

Изучить парадигмы программирования, в частности ООП. Создать систему классов для градостроительной компании.

**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: # 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().

**Основные теоретические положения.**

Объектно-ориентированная парадигма

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.

Основные понятия:

- Объект - конкретная сущность предметной области.
- Класс - тип объекта. Классы содержат атрибуты, которые подразделяются на поля и методы.
- Поле - это переменная, которая определена внутри класса.
- Под методом понимают функцию, которая определена внутри класса.
    - Конструктор - это специальный метод, который нужен для создания объектов класса.

Инструкция raise

Исключения - это специальный класс объектов в языке Python. Исключения предназначены для управления теми участками программного кода, где может возникнуть ошибка.

Для самостоятельно генерирования исключений применяется инструкция raise.

Синтаксис: raise <Создание объекта исключения>

Инструкция raise генерирует исключение, которое должно быть объектом класса, являющегося наследником класса Exception.

Lambda-выражения

Лямбда-выражения - это специальный элемент синтаксиса для создания анонимных (т.е. без имени) функций по месту их использования. Используя лямбда-выражения можно объявлять функции в любом месте кода, в том числе внутри других функций.

Синтаксис:

lambda аргумент1, аргумент2,..., аргументN : выражение

Функция filter

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

Функция <функция> применяется для каждого элемента итерируемого объекта <*объект*> .

**Выполнение работы.**

1) Класс *HouseScheme*:

   Класс-родитель.

   ❖ Поля класса:

- *self.rooms* - количество жилых комнат.
- *self.area* - площадь (в квадратных метрах, не может быть отрицательной).
- *self.restroom* - совмещенный санузел (значениями могут быть или False, или True).

Реализована проверка переданных в конструктор параметров, при несоответствии требованиям - выбрасывается исключение ValueError с текстом → *'Invalid value'*.

2) Класс *CountryHouse*:

Наследуется от класса *HouseScheme*.

❖ Поля класса:

- *self.storey* - количество этажей (не может быть отрицательным).
- *self.ter_area* - площадь участка (не может быть отрицательной).

Реализована проверка переданных в конструктор параметров, при несоответствии требованиям - выбрасывается исключение ValueError с текстом → *'Invalid value'*.

✓ Метод *__str__* - метод, который возвращает преобразованную к нужному виду строку с необходимой информацией об объекте.

✓ Метод *__eq__* - метод, который возвращает True или False при сравнении двух объектов типа CountryHouse.

3) Класс *Apartment:*

Наследуется от класса *HouseScheme*.

❖ Поля класса:

- *self.storey* - этаж (может быть число от 1 до 15).
- *self.window* - куда выходят окна (значением может быть одна из строк: N, S, W, E).

Реализована проверка переданных в конструктор параметров, при несоответствии требованиям - выбрасывается исключение ValueError с текстом → *'Invalid value'*.

- ✓ Метод *__str__* - метод, который возвращает преобразованную к нужному виду строку с необходимой информацией об объекте.

4) Класс *CountryHouseList:*

Наследуется от класса *list.*

- ❖ Поле класса:
  - *self.name* - имя списка.
- ✓ Метод *append* - метод, который добавляет подходящий (если p_object - деревенский дом) элемент в конец списка, иначе выбрасывает исключение TypeError с текстом: '*Invalid type <тип_объекта p_object>'.*
- ✓ Метод *total_square* - метод, который посчитывает и возвращает общую жилую площадь.

5) Класс *ApartmentList:*

Наследуется от класса *list.*

- ❖ Поле класса:
  - *self.name* - имя списка.
- ✓ Метод *extend* - метод, который добавляет подходящий (если элемент *iterable* - объект класса Apartment) элемент, иначе - элемент игнорируется. Проверка реализована через лямбда-выражение.
- ✓ Метод *floor_view* - метод, который выводит квартиры в необходимом формате, этаж (*storey*) которых входит в переданный диапазон (*floors*) и окна (*window*), направления которых выходят в одном из переданных направлений (*directions*). Проверка реализована через лямбда-выражение и функциию *filter().*

- ▸ Иерархия классов:
  - Родитель: *HouseScheme*

    Наследники*: CountryHouse, Apartment.*
  - Родитель*: list.*

    Наследники: *CountryHouseList, ApartmentList.*

- Переопределённые Методы:
  - *__init__()*
  - *__str__()*
  - *__eq__()*
  - *append()*
  - *extend()*
- Метод __str__() будет вызван в случаях, когда будет необходимо преобразование объекта к типу str (к примеру, для возвращения или вывода).
- Непереопределенные методы класса list для CountryHouseList и ApartmentList будут работать, так как данные классы являются наследниками класса-родителя list → все методы list'а будут работать и для классов-наследников. Например: self.sort() - отсортирует список, self.remove() - развернёт список и т.д.

Разработанный программный код см. в приложении А.

**Тестирование.**

Код 1 (доп).

```
test_1_CH=CountryHouse(5,50,True,3,100)
test_2_CH=CountryHouse(5,50,True,3,100)
test_3_CH=CountryHouse(5,50,False,3,100)
test_list_CH=CountryHouseList("CountryHouse")
test_list_CH.append(test_1_CH)
test_list_CH.append(test_2_CH)
test_list_CH.append(test_3_CH)
print(test_1_CH)
print(test_1_CH == test_2_CH)
print(test_1_CH == test_3_CH)
print(test_list_CH)
print(test_list_CH.total_square())
```

Код 2 (доп).

```
test_n_CH=CountryHouse(5,50,True,-1,100)
print(test_n_CH)
```

Код 3 (доп).

```
test_1_A=Apartment(5,50,True,3,'N')
test_2_A=Apartment(5,50,True,3,'W')
test_list_A=ApartmentList("Apartment")
test_list_A.extend([test_1_A, test_2_A, 'testNone'])
print(test_1_A)
print(test_list_A)
print(test_list_A.floor_view([1,7],['W', 'E']))
```

Код 4 (доп).

```
test_n_A= Apartment(5,50,True,1,'D')
print(test_n_A)
```

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | Код 1. | Country House: Количество жилых комнат 5, Жилая площадь 50, Совмещенный санузел True, Количество этажей 3, Площадь участка 100.<br>True<br>True<br>[<__main__.CountryHouse object at 0x03C21A70>,<__main__.CountryHouse object at 0x03C21AB0>,<__main__.CountryHouse object at 0x03C21AD0>]<br>150 | Ответ верный. |
| 2. | Код 2. | ValueError: Invalid value | Ответ верный. |

| | | | |
|---|---|---|---|
| 3. | Код 3. | Apartment: Количество жилых комнат 5, Жилая площадь 50, Совмещенный санузел True, Этаж 3, Окна выходят на N.<br>[<__main__.Apartment object at 0x03440AF0>,<br><__main__.Apartment object at 0x03440B30>]<br>W: 3 | Ответ верный. |
| 4. | Код 4. | ValueError: Invalid value | Ответ верный. |

**Выводы.**

Были изучены парадигмы программирования, в частности ООП и создана система классов для градостроительной компании.

Разработана программа, представляющая собой систему классов для градостроительной компании: предоставлена иерархия классов, инициализированы поля классов и переопределены все необходимые методы. Реализована проверка поступающих значений заданным требованиям через исключения (инструкцию raise). Для проверки условий используются лямбда-выражения и функция filter.

.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.py

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

class CountryHouse(HouseScheme):
    def __init__(self,rooms,area,restroom,storey,ter_area):
        super().__init__(rooms,area,restroom)
        if storey>=0 and ter_area>=0:
            self.storey = storey
            self.ter_area = ter_area
        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.area,self.restroom,self.storey,self.ter_area)
        def __eq__(self, new_house):
            return new_house.area==self.area and new_house.ter_area==self.ter_area and -1<=(new_house.storey-self.storey)<=1
```

```
class Apartment(HouseScheme):
    def __init__(self,rooms,area,restroom,storey,window):
        super().__init__(rooms,area,restroom)
        if 1<=storey<= 15 and (window in ['N','S','W','E']):
            self.storey = storey
            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.area,self.restroom,self.storey,self.window)

class CountryHouseList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name
    def append(self, p_object):
        if type(p_object)==CountryHouse:
            super().append(p_object)
        else:
            raise TypeError('Invalid type {}'.format(type(p_object)))
    def total_square(self):
        S = 0
        for item in self:
            S = S+item.area
        return S
```

```
class ApartmentList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name
    def extend(self,iterable):
        super().extend(filter(lambda x: type(x)==Apartment,iterable))
    def floor_view(self,floors,directions):
        result = list(filter(lambda x: (x.window in directions) and floors[0]<=x.storey<=floors[1], self))
        for item in result:
            print('{}: {}'.format(item.window,item.storey))
```