

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра РЭС

ОТЧЕТ

по учебной практике

**Тема: Рефакторинг кода. Проектирование встроенных приложений умного
дома**

Студент гр. 1181

Лысенко А.В.

Руководитель

Проценко И.М.

Санкт-Петербург

2023

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Лысенко А.В.

Группа 1181

Тема практики: Рефакторинг кода. Проектирование встроенных приложений умного дома

Задание на практику:

Произвести рефакторинг (упрощение, расширение и понимание без изменения поведения) кода, а именно:

- a) Введение новых классов и функций
- b) Обоснование введения новых классов и функций
- c) Uml диаграмма классов
- d) Функциональная схема программы
- e) Комментирование новых частей кода

Сроки прохождения практики: 01.02.2023 – 22.05.2023

Дата сдачи отчета: 09.05.2023

Дата защиты отчета: 09.05.2023

Студент		Лысенко А.В.
Руководитель		Проценко И.М.

АННОТАЦИЯ

Целью данной работы является развитие профессиональных навыков в понимании и обработки чужого программного кода. В данной работе произведено:

- a) Введение новых классов и функций
- b) Обоснование введения новых классов и функций
- c) Uml диаграмма классов
- d) Функциональная схема программы

Благодаря этой работе был улучшен программный код учебного комплекса для проектирования устройств умного дома.

SUMMARY

The purpose of this work is to develop professional skills in understanding and processing someone else's program code. This work produced:

- a) Introduction of new classes and functions
- b) Rationale for installing new classes and functions
- c) Uml class diagram
- d) Functional scheme of the program

Thanks to this work, the program code of the educational complex for designing smart home devices has been improved.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1. ВВЕДЕНИЕ НОВЫХ КЛАССОВ И ФУНКЦИЙ.....	6
1.1. Функции	6
1.2. Классы и методы	6
2. UML ДИАГРАММА КЛАССОВ.....	11
3. ФУНКЦИОНАЛЬНАЯ СХЕМА ПРОГРАММЫ	13
ЗАКЛЮЧЕНИЕ	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	16

ВВЕДЕНИЕ

Целью практического задания является изменение вида программного кода и его оптимизация, с возможностью дальнейшей эксплуатации и улучшения. Для этого была использована парадигма ООП, благодаря которой был введен класса, который наследует элементы других классов. Также была улучшена читаемость кода с помощью применения стандарта PEP-8 и сопутствующего комментирования.

1. ВВЕДЕНИЕ НОВЫХ КЛАССОВ И ФУНКЦИЙ

1.1. Функции

Функция `read_conf()` - открывает файл 'config.json', загружает его содержимое в переменную 'conf' в формате словаря (dictionary) при помощи функции 'json.load()', а затем выводит все ключи словаря 'conf' при помощи цикла 'for'. Функция необходима для работы класса, который будет в дальнейшем описан.

```
def read_conf() -> dict:
    f = open('config.json')
    conf = json.load(f)
    f.close()
    print("Found arguments:")
    for i in conf:
        print(i)
    return conf
```

1.2. Класс и его методы

В коде, предоставленном из задания, подавляющее большинство функций получилось объединить в методы единственного класса `AppWindow()`, так как было возможно описать их взаимодействие внутри лишь одного класса. Сам же класс является наследником класса `QMainWindow` из библиотеки `PyQt5`, однако, стоит заметить, что класс `QMainWindow` является наследником класса `QWidget`. Также он имеет атрибуты, которые представлены на UML диаграмме, в основном это словари и атрибуты других классов.

Инициализатор - это конструктор, который вызывается, когда создается экземпляр класса. Сам же `__init__` является динамическим полем класса.

```
def __init__(self, conf: dict):
    super(AppWindow, self).__init__()
```

Где параметр `self` является ссылкой на объект класса, во время его создания. `Conf dict` – это вышеупомянутый словарь, созданный из json данных. Далее используется функция `super()` для обращения к базовому классу и сохранения иерархии наследования. Данная

функция возвращается ссылкой на объект-посредник, через который проходит вызов базового класса.

```
Ui_MainWindow = uic.loadUi(os.path.join(conf['uiPath'],  
conf['uiFileName']), self)
```

Далее происходит парсинг файла с расширением .ui для получения ссылки на объект формы, сконвертированный из формы диалейнера в питон объект.

```
self.nam = QNetworkAccessManager()
```

nam экземпляр класса qnetworkaccessmanager().

```
self.plot = Plot(Ui_MainWindow.plotwidget)
```

plot экземпляр класса Plot().

```
self.setWindowTitle("Lr4")
```

Выставление надписи на рабочем окне проекта.

```
Ui_MainWindow.lineEdit_URL.setText("http://" +  
conf['defaultMDNSname'] + conf['defaultPostRoute'])
```

Установка URL адреса исходя из полученных значений от словаря conf.

```
for i in range(1, 4):
```

```
    getattr(Ui_MainWindow,  
f"pushButton_switch_lamp{i}").setCheckable(True)  
    getattr(Ui_MainWindow,  
f"pushButton_switch_lamp{i}").setChecked(False)  
    getattr(Ui_MainWindow, f"label_lamp_on{i}").hide()  
    getattr(Ui_MainWindow,  
f"pushButton_switch_lamp{i}").toggled["bool"].connect(lambda val,  
i=i: self.handle_toggle_lamp(i, val))
```

В данном цикле происходит инициализация режимов лампочек и их видимость, за счет getattr()-встроенной функции, возвращающая значения атрибута объекта. Причем такая инициализация возможна благодаря индексам переименованных действий над лампочками.

```
Ui_MainWindow.pushButton_send_post.clicked.connect(self.send_message)
```

Привязка метода send_message к кнопке «Отправить»

```
Ui_MainWindow.pushButton_send_get.clicked.connect(self.get_value_from_mocket)
```

Привязка метода send_value_from_mocket к кнопке «Отправить GET запрос»

```
Ui_MainWindow.led_array = [getattr(Ui_MainWindow, f"leds{i}")  
for i in range(1,9)]
```

Формирование списка для хранения данных о RGB ленте

```
self.rgb_leds_state = conf["defaultRGBLeds"]
```

Атрибут для хранения исходных значений светодиодов

```
self.switch_all(False)
```

Выключение светодиодов с помощью метода `switch_all`, данный метод будет описан в дальнейшем.

```
for led in Ui_MainWindow.led_array:
```

```
    led.mousePressEvent = self.set_color
```

Изменение цвета светодиода по нажатию мышки.

```
Ui_MainWindow.pushButton_leds_on.clicked.connect(lambda:
```

```
self.switch_all(True))
```

Обработка кнопки включения индикаторов.

```
Ui_MainWindow.pushButton_leds_off.clicked.connect(lambda:
```

```
self.switch_all(False))
```

Обработка кнопки выключения индикаторов.

```
Ui_MainWindow.pushButton_leds_color.clicked.connect(lambda:
```

```
self.set_color_all())
```

Обработка кнопки изменения цвета индикаторов.

```
self.timer = QTimer(self)
```

```
    self.timer.setInterval(conf["defaultUpdateInterval"])
```

```
    self.timer.timeout.connect(self.get_value_from_mocket)
```

Подключение таймера и выборки данных с интервалом от значений по умолчанию.

```
Ui_MainWindow.spinBox_autoupdate.setValue(conf["defaultUpdateInterval"] // 1000)
```

Инициализация интервала авто обновления на входе.

Описание методов класса.

```
def handle_toggle_lamp(self, n: int, checked: bool):
```

Метод переключения состояния определенной лампочки. Для этого этот метод принимает номер лампы, соответствия с ним на экране появится интересующая лампа и надпись «Выкл» или «Вкл», в зависимости от параметра `checked(True/False)`.

```
def switch_all(self, on: bool):
```

Метод созданный из двух функций `vk1` и `vik1` исходного кода.

Отвечает за включение и выключение всех светодиодов за счет переприсвоения цветовых параметров, исход зависит от

принимаемого параметра `on`. Если `on` – `True`, тогда все лампы включены, иначе выключены.

def set_color_all(self):

Метод устанавливающий цвета светодиодов за счет данных получаемых от пользователя.

def set_color(self, event: QMouseEvent):

Метод для установки определенного цвета определенного светодиода за счет запроса из диалога по щелчку мыши.

def paint_led_color(self, led: QLabel, color: QColor):

Метод необходимый для работы **def set_color(self, event: QMouseEvent):**, именно с помощью него выставляется необходимый цвет у некоторого светодиода.

def collect_lamps_state(self) -> dict[str, bool]:

Метод для создания словаря, описывающего состояние всех ламп.

def update_buttons(self, bs: list[bool]):

Метод для переключения тумблеров на панели и изменения их видимости. Состояние тумблера определяется исходя из соответствующего элемента принимаемого списка.

def compose_post_json_data(self) -> dict:

Формирование словаря из состояний лампочек и светодиодов для дальнейшей отправки.

def convert_buttons_state(self, data: dict[str, bool]) -> list[bool]:

Метод для формирования списка о состоянии кнопок, так как состояние кнопки дискретно, то и список будет состоять из элементов `True/False`.

def update_lcds(self, data: dict):

Метод для изменения цвета светодиодов с помощью полученного словаря и встроенного списка с необходимыми цветами.

def update_colors(self, new_state: dict[str, dict[int]]):

Метод для обновления цветов светодиодов за счет метода `paint_led_color` и изменения значений внутри исходного словаря `defaultRGBLEds` из файла `conf`. Данный метод принимает словарь, значение которого является списком, и за счет метода `paint_led_color` изменяет значения в принятом словаре.

Вывод: изменение функций и объединение их под одним классом было необходимо для: более удобного взаимодействия между ним, улучшения читаемости и осмысления кода, более удобного использования в дальнейшей эксплуатации кода, возможности более простого расширения функционала кода, упрощенного редактирования. В дальнейшем полученный класс и его методы можно будет вынести в отдельный файл, который можно будет импортировать в файл main.py и в нем уже продолжить работу.

2. UML ДИАГРАММА КЛАССОВ

Это структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними. Благодаря ней можно более быстро и четко понять каким образом взаимодействуют классы внутри проекта; какие атрибуты имеют, и как выглядит их поле; какие от них получаются экземпляры; также указываются все методы. На Рисунке 1 представлена диаграмма классов для проекта, полученного из задания.

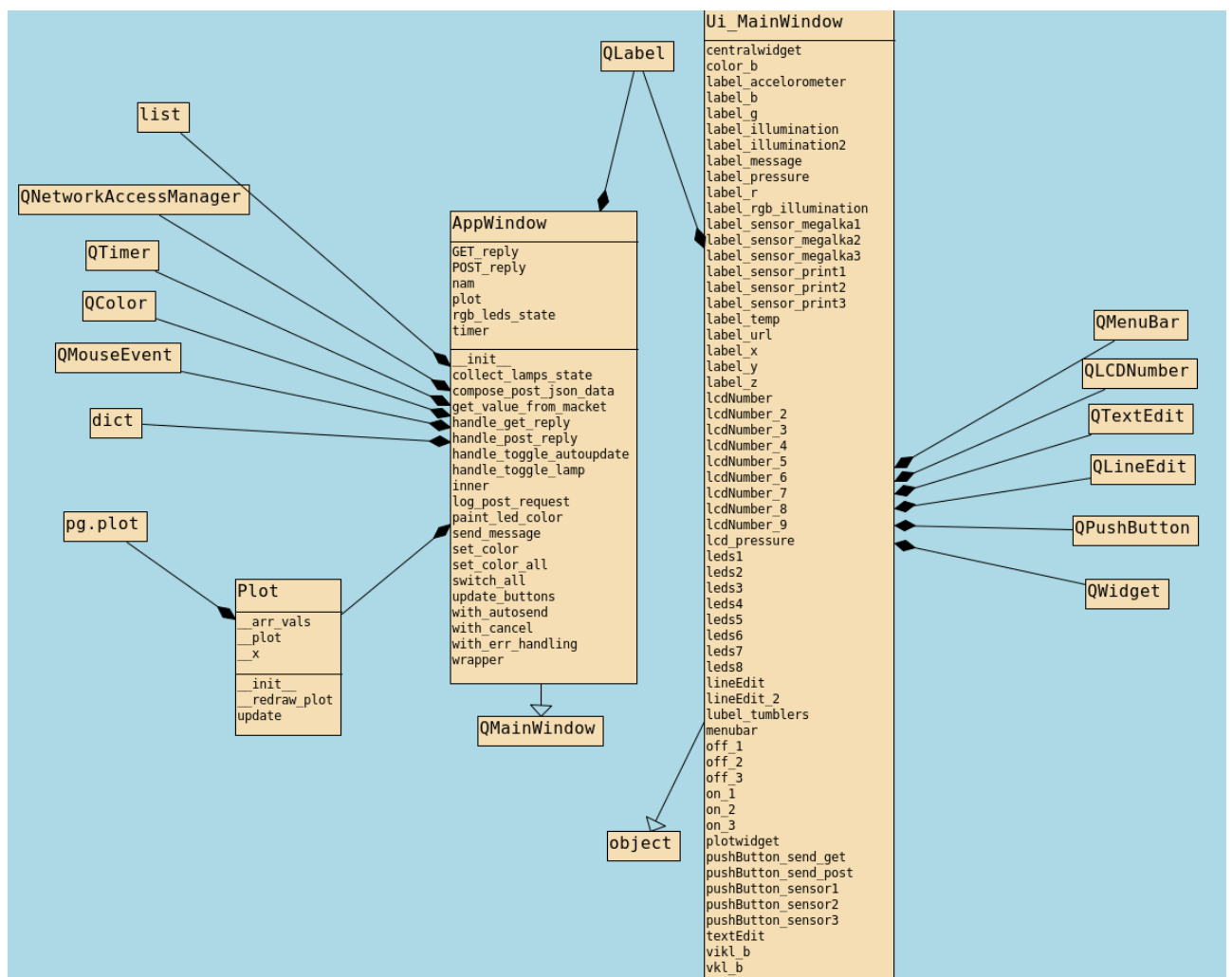


Рисунок 1 –UML диаграмма

Вывод: в результате получилась наглядная диаграмма с несколькими классами, которая показывает: их название, содержимое их полей, каким образом происходит наследование, какие экземпляры участвуют в работе программы,

какие методы существуют. Данная диаграмма поможет быстро понять, как взаимодействуют классы, без просмотра и разбора программного кода.

3. ФУНКЦИОНАЛЬНАЯ СХЕМА ПРОГРАММЫ

Функциональная схема или схема данных (ГОСТ 19.701-90) - схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.


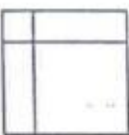






Название блока	Обозначение	Назначение блока
Запоминаемые данные		Для обозначения таблиц и других структур данных, которые должны быть сохранены без уточнения типа устройства
Оперативное запоминающее устройство		Для обозначения таблиц и других структур данных, хранящихся в оперативной памяти
Запоминающее устройство с последовательной выборкой		Для обозначения таблиц и других структур данных, хранящихся на устройствах с последовательной выборкой (магнитной ленте и т.п.)
Запоминающее устройство с прямым доступом		Для обозначения таблиц и других структур данных, хранящихся на устройствах с прямым доступом (дисках)
Документ		Для обозначения таблиц и других структур данных, выводимых на печатающее устройство
Ручной ввод		Для обозначения ручного ввода данных с клавиатуры
Карта		Для обозначения данных на магнитных или перфорированных картах
Дисплей		Для обозначения данных, выводимых на дисплей компьютера

Рисунок 2— обозначения для функциональной схемы согласно ГОСТ 19.701-90

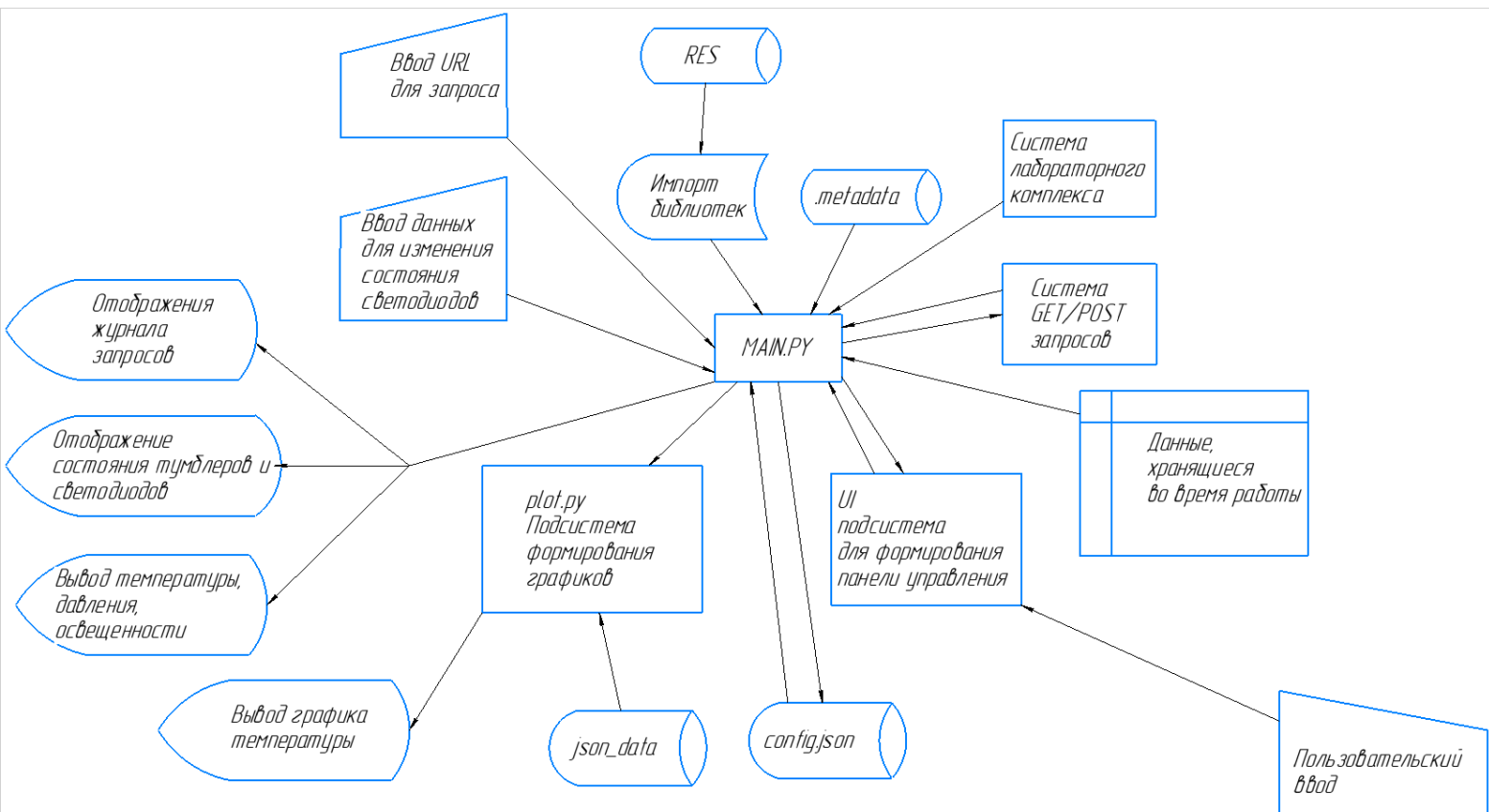


Рисунок 3– функциональная схема программы

Вывод: функциональная схема позволяет увидеть взаимодействие элементов программы, без просмотра программного кода и дополнительных файлов.

ЗАКЛЮЧЕНИЕ

Результатом работы является полученный переработанный программный код, написан в соответствии с парадигмой ООП, а именно был составлен класс, в котором по смыслу были объединены исходные функции. Данное решение позволило упростить их взаимодействие, убрать некоторые нагромождение в виде дублирования ветвлений if/else и лишних функций, которые были объединены. Также был применен стандарт PEP-8, в соответствии с ним программа получила следующий структурный вид: начало- импорт библиотек и исходных файлов, объявление глобальных констант; после- объявление и формирование тела класса с необходимыми методами; в завершении – тело программы. В добавок согласно стандарту имя класса – существительное, написанное с большой буквы; имена методов/функций – глагол с некоторым существительным, написанный с маленькой буквы, для простоты понимания действия; глобальные константы, атрибуты класса и переменные – существительные с маленькой буквы. В случае, если название состоит из нескольких слов использовалось нижнее подчеркивание. Максимальная длина строки не более 72 символов. Во всем коде сохраняется табуляция. По необходимости в некоторых местах были сделаны сквозные блоки комментарии с соответствующим оформлением. При объявлении функции был указан тип аргумента, так же указывается тип и название возвращаемого значения, если оно есть. Соблюдение данного стандарта помогает улучшить читаемость кода и уменьшает время на его понимание. Благодаря полученным схемам из пунктов 2 и 3 пользователь данной программы сможет быстро понять её работу, не прибегая к чтению какой-либо документации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Python для модуля threading // Python 3.11.3 documentation. URL: <https://docs.python.org/3/library/asyncio.html> (дата обращения 07.06.2023).
2. Документация Python для модуля asyncio // Python 3.11.3 documentation. URL: <https://docs.python.org/3/library/threading.html> (дата обращения 07.06.2023).
3. Руководство для Python для стандарта PEP-8 // Python 3.11.3 documentation. URL: <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html> (дата обращения 07.06.2023).
4. Руководство для Python для парадигмы ООП // Python 3.11.3 documentation. URL: <https://pythonworld.ru/osnovy/obektno-orientirovannoe-programmirovanie-obshhee-predstavlenie.html> (дата обращения 07.06.2023).