МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: Поиск дубликатов файлов

Студент гр. 8309 _____ Ворончихин А.А.

Преподаватель _____ Красильников А.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Ворончихин А.А.

Группа 8309

Тема работы: Поиск дубликатов файлов

Исходные данные:

Требуется создать программу с графическим интерфейсом, производящую поиск одинаковых файлов и их удаление.

Содержание пояснительной записки:

Пояснительная записка содержит: введение, описание хода работы, демонстрацию работы, заключение и листинг с кодом на языке Python 3.9

Предполагаемый объем пояснительной записки:
Не менее 5 страниц.

Дата выдачи задания: 01.11.2020

Дата сдачи реферата: 06.03.2021

Дата защиты реферата: 06.03.2021

Студент _____ Ворончихин А.А.

Преподаватель _____ Красильников А.В.

АННОТАЦИЯ

В курсовой работе содержится решение задачи и демонстрация работы. В ходе выполнения данной работы были получены важные практические навыки и закреплены теоретические знания материала. Результаты, которые были получены были проверены с помощью различных файлов, что подтверждает их работоспособность.

SUMMARY

The course work contains a solution to the problem and a demonstration of the work. In the course of this work, important practical skills were obtained and theoretical knowledge of the material was consolidated. The results that were obtained were checked using various files, which confirms their operability.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: написать приложение с графическим интерфейсом для поиска повторяющихся файлов в выбранной директории. Пользователь выбирает через диалог желаемую директорию. Программа осуществляет поиск всех возможных дубликатов файлов (по содержимому, а не по имени файла, конечно же) в этой директории и во всех вложенных и показывает пользователю результат. Должна быть предусмотрена возможность удаления дубликатов через интерфейс программы

Основная задача: написать программу на языке программирования Python 3.9 с применением графического интерфейса

1. ХОД РАБОТЫ

1.1. Применяемые языки, модули и библиотеки

В работе применялся язык программирования Python актуальной версии 3.9.2. При разработке применялись следующие модули/библиотеки:

Таблица 1 – Применяемые модули.

№	Модуль	Назначение
1	hashlib	Модуль предназначен для создания hash сравниваемых файлов. Hash разных файлов всегда отличается
2	os	Модуль предназначен для взаимодействия с операционной системой
3	sys	Модуль обеспечивает доступ к функциям интерпретатора, например, поиск путей объектов
4	PyQt5	Модуль предназначен для создания графического интерфейса. Вместе с ним применялся Qt Designer – конструктор интерфейсов
5	WPF*	Модуль предназначен для создания графического интерфейса
6	Windows Forms*	Модуль предназначен для создания графического интерфейса
7	__future__*	Модуль позволяет импортировать функционал более новых версий ЯП Python

Примечание: под символом «*» отмечены модули, не попавшие в финальную версию программы. Отказ от WPF и Windows Forms связан с тем, что для их реализации используется версия ЯП IronPython, являющаяся модификацией синтаксиса Python версии 2.7. В связи с тем, что современная версия языка 3.9, код, который был написан к реализации интерфейса, не был совместим с этими модулями и с IronPython в целом. Модуль __future__ не смог повлиять на ситуацию, поэтому, в последний момент было принято решение перейти на модуль PyQt5, как на более современный и поддерживающийся.

Были применены IDE Microsoft Visual Studio с пакетами Python и IronPython, после PyCharm и Qt Designer

1.2. Этапы проектирования курсовой работы

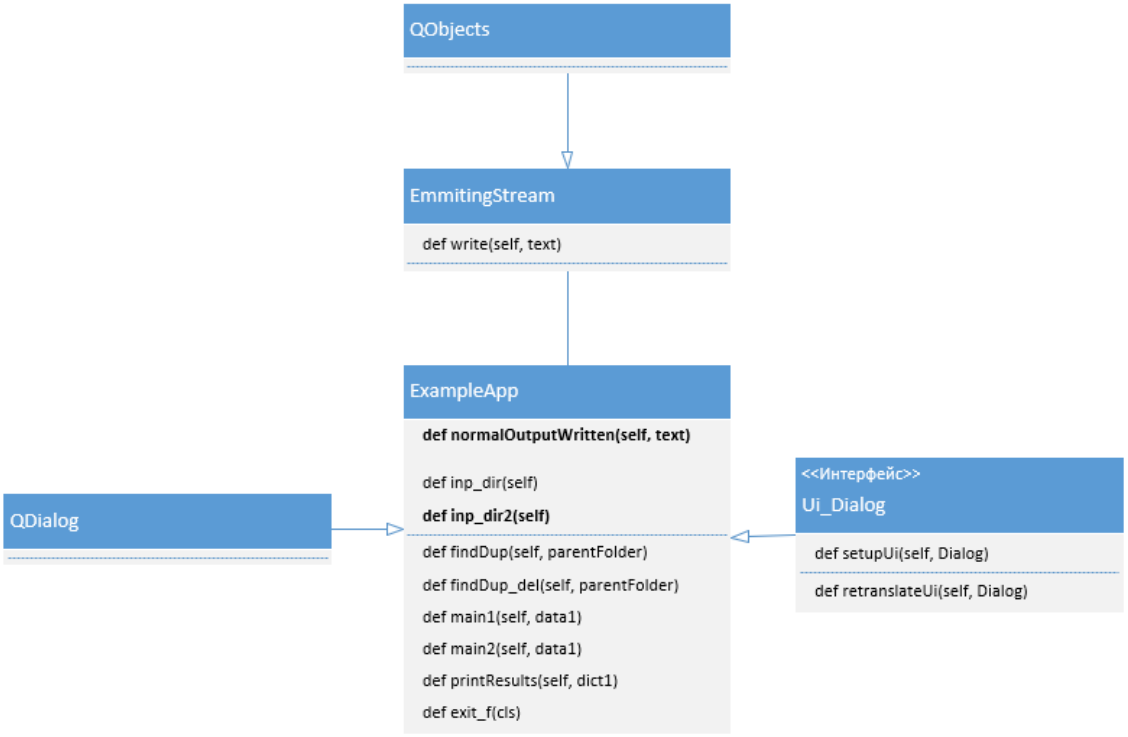

Рисунок 1 – Диаграмма классов UML

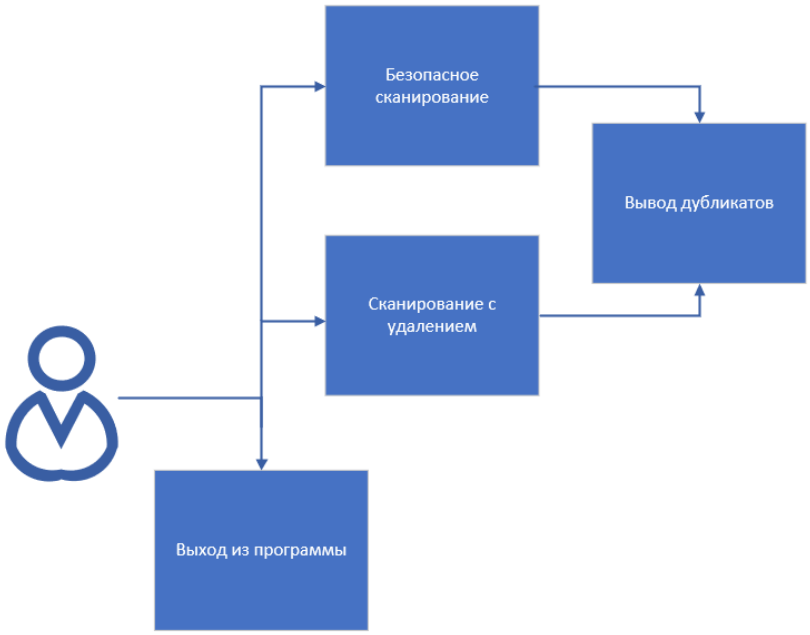

Рисунок 2 – Use – Case диаграмма

2. ЛИСТИНГ ПРОЕКТА

2.1. inter3.py

```
# -*- coding: utf-8 -*-
# Form implementation generated from reading ui file
'D:\courseoop\inter2.ui'
# Created by: PyQt5 UI code generator 5.15.3
# WARNING: Any manual changes made to this file will be lost when
pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(470, 419)
        self.SearchButton = QtWidgets.QPushButton(Dialog)
        self.SearchButton.setGeometry(QtCore.QRect(10, 10, 111, 41))
        self.SearchButton.setObjectName("SearchButton")
        self.textEdit = QtWidgets.QTextEdit(Dialog)
        self.textEdit.setGeometry(QtCore.QRect(130, 10, 331, 401))
        self.textEdit.setObjectName("textEdit")
        self.DeleteButton = QtWidgets.QPushButton(Dialog)
        self.DeleteButton.setGeometry(QtCore.QRect(10, 60, 111, 41))
        self.DeleteButton.setObjectName("DeleteButton")
        self.ExitButton = QtWidgets.QPushButton(Dialog)
        self.ExitButton.setGeometry(QtCore.QRect(10, 370, 111, 41))
        self.ExitButton.setObjectName("ExitButton")
        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dublicate Search
1.4 Release"))
        self.SearchButton.setText(_translate("Dialog", "Search"))
        self.DeleteButton.setText(_translate("Dialog",
"Search&Delete"))
        self.ExitButton.setText(_translate("Dialog", "Exit"))

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

```
# -------------------------------- from workcode.py ------------------
    @classmethod
    def hashfile(self, path, blocksize = 65536):
        afile = open(path, 'rb')
        hasher = hashlib.md5()
        buf = afile.read(blocksize)
        while len(buf) > 0:
            hasher.update(buf)
            buf = afile.read(blocksize)
        afile.close()
        return hasher.hexdigest()

    @classmethod
    def findDup(self, parentFolder):
        dups = {}
        for dirName, subdirs, fileList in os.walk(parentFolder):
            for filename in fileList:
                path = os.path.join(dirName, filename)  # Полный путь к
файлу
                file_hash = self.hashfile(path) # Вычисление хэша
                if file_hash in dups:
                    dups[file_hash].append(path) # добавление дубликата
                else:
                    dups[file_hash] = [path]    # добавление нового
типа дубликата
        return dups

    @classmethod
    def findDup_del(self, parentFolder):
        dups = {}
        for dirName, subdirs, fileList in os.walk(parentFolder):
            for filename in fileList:
                path = os.path.join(dirName, filename)  # Полный путь к
файлу
                file_hash = self.hashfile(path)  # Вычисление хэша
                if file_hash in dups:
                    dups[file_hash].append(path)  # добавление дубли-
ката
                    os.remove(path)
                else:
                    dups[file_hash] = [path]  # добавление нового типа
дубликата
        return dups

    @classmethod
    def joinDicts(self, dict1, dict2):
        for key in dict2.keys():
            if key in dict1:
                dict1[key] = dict1[key] + dict2[key]
            else:
                dict1[key] = dict2[key]
```

```
@classmethod
    def main1(self, data1):
        dups = {}
        self.joinDicts(dups, self.findDup(data1))
        self.printResults(dups)

    @classmethod
    def main2(self, data1):
        dups = {}
        self.joinDicts(dups, self.findDup_del(data1))
        self.printResults(dups)

    @classmethod
    def printResults(self, dict1):
        results = list(filter(lambda x: len(x) > 1, dict1.values()))
        if len(results) > 0:
            print('Duplicates Found:')
            print('The following files are identical. The name could
differ, but the content is identical')
            print('____________________')
            for result in results:
                for subresult in result:
                    print(subresult)
                print('____________________')
        else:
            print('No duplicate files found.')

    @classmethod
    def exit_f(cls):
        sys.exit()

# ------------------------------------------------------------------------
-------------------------------------


def main():
    #global status #
<--- обьявление переменной
    #status = False
    app = QtWidgets.QApplication(sys.argv)  # Новый экземпляр
QApplication
    window = ExampleApp()  # Создаём объект класса ExampleApp
    window.show()  # Показываем окно
    app.exec_()  # и запускаем приложение


if __name__ == '__main__':
    main()
```

Для запуска программы требуется создать проект из этих файлов в одной директории, при этом необходимо, чтобы к IDE был подключен PyQt. Установщики PyQt можно легко найти в интернете, однако, установка иногда различается. Например, к PyCharm нельзя присоединить этот модуль через стандартные инструкции с применением командной строки. Дополнительно нужно в настройках PyCharm выбрать Tools> External Tools, там с помощью кнопки следует добавить пакеты PyQt.

В ходе выполнения проекта было замечено, что PyQt в отличие от Qt имеет ряд ограничений и особенностей, не позволяющих использовать функционал Qt на полную мощность. Основной принцип также основан на системе сигналов-слотов, однако, PyQt не позволяет ее использовать с тем удобством, с каким ее можно использовать в Qt с языком C++. Так, например, оказалось невозможным интегрировать в рабочий алгоритм элементы CheckBox, ListWidget, так как булевы значения этих функций не обнаруживались при вызове связанных с ними методов даже при пересохранении в отдельную переменную. Предположительно, после применения слота все созданные внутри атрибуты класса уничтожаются.

В итоге было создано приложение со следующим интерфейсом:

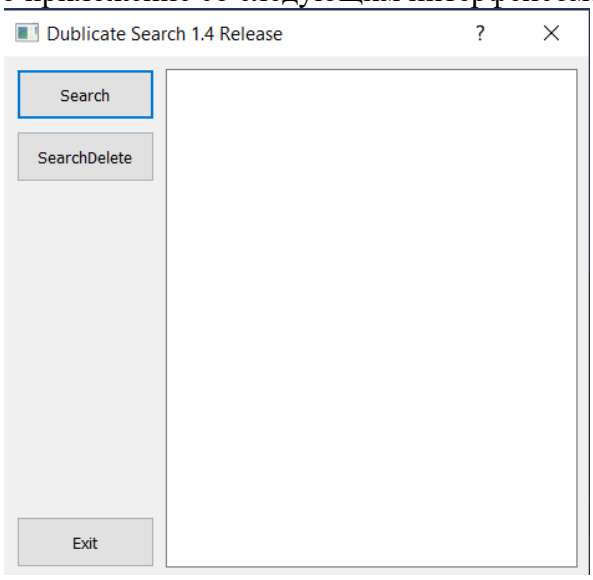

Рисунок 3 – Интерфейс программы

Для удобства пользователя было реализовано диалоговое окно выбора директории:

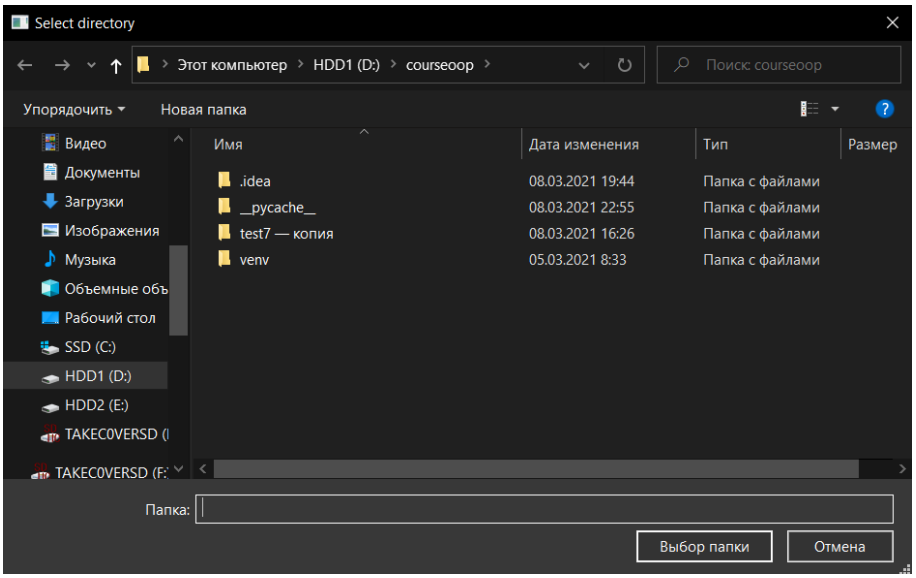

Рисунок 4 – Выбор директории

Так как применение элементов, специально созданных для выбора вариантов, оказалось невозможным, выбор между безопасным сканированием и автоматическим удалением происходит путем выбора соответствующих кнопок. Также предусмотрена процедура выхода из программы с использованием кнопки выхода, так как при закрытии окна интерфейса с помощью кнопки сверху вызывало ошибку, связанную с объектами на языке C++ в PyQt.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были улучшены навыки владения языком Python 3.9, а также среды создания графических приложений WPF на основе IronPython и PyQt на основе Python 3. Однако, было выяснено, что язык Python не очень хорошо подходит для использования в данной дисциплине, так как объектно-ориентированное программирование встречается со сложностями при реализации графического интерфейса. Модули для реализации GUI либо устарели, как IronPython WPF, либо не предоставляют полного функционала, как PyQt. Однако все негативные моменты также были усвоены.