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

Практическая работа №2

по дисциплине

«Объектно-ориентированное программирование»

Студент гр. 4351 _____ Чурзин Г.В.

Преподаватель _____ Кулагин М.В.

Санкт-Петербург
2025

Задание

Даны 2 файла-справочника городов (Файлы к ПР2). Один файл в формате xml, другой в формате csv.

Необходимо разработать консольное приложение для работы с ними.

После запуска приложение ожидает ввода пути до файла-справочника либо команды на завершение работы (какая-то комбинация клавиш).

По команде завершения работы приложение завершает свою работу.

После ввода пути до файла-справочника приложение формирует сводную статистику:

1) Отображает дублирующиеся записи с количеством повторений.

2) Отображает, сколько в каждом городе: 1, 2, 3, 4 и 5 этажных зданий.

3) Показывает время обработки файла.

После вывода статистики приложение снова ожидает ввода пути до файла-справочника либо команды на завершение работы.

В процессе работы приложение падать не должно, выход только по команде на завершение работы.

Спецификация программы

Класс *Searcher*:

Методы:

1. run(self) обеспечивает зацикленность выбора файла и запуска всех функций обработки

Класс *FileParsing*:

Поля:

1. file_path — путь к файлу с адресами
2. addresses — список объектов адресов

Методы:

1. __init__(self, file_path) инициализирует путь к файлу и список адресов
2. parse(self) определяет способ парсинга
3. parse_csv(self) производит парсинг csv файла, возвращая список адресов
4. parse_xml(self) производит парсинг xml файла, возвращая список адресов

Класс *Address*:

Поля:

1. city — название города
2. street — название улицы
3. house — номер дома
4. floor — количество этажей

Методы:

1. __init__(self, city, street, house, floor) инициализирует параметры адреса
2. object(self) возвращает все параметры адреса одним объектом для поиска дубликатов

Класс *Statistics*:

Поля:

1. addresses — список полученных из файла адресов

Методы:

1. __init__(self, addresses) инициализирует список адресов для обработки
2. find_duplicates(self) обеспечивает поиск и возврат списка дубликатов
3. floor_statistics(self) собирает статистику этажей по городам и возвращает двумерный массив
4. print_search_results(self, duplicates, floor_stats) выводит дубликаты и этажи городов в терминал.

UML-диаграмма классов представлена на рисунке 1.

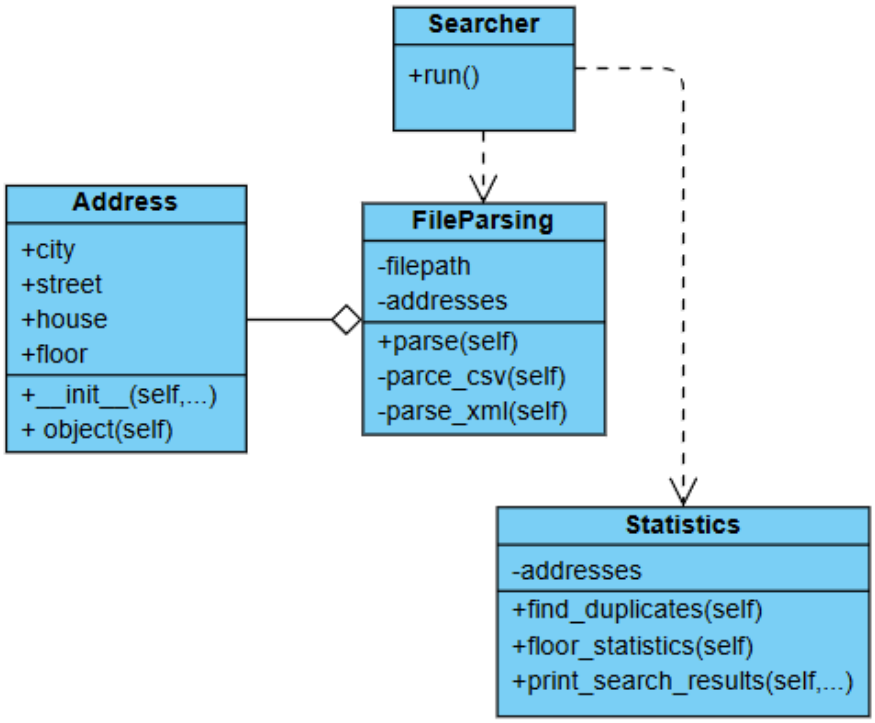

Рисунок 1

Описание интерфейса пользователя программы

Примеры работы приведены на рисунках 2 — 5. Вначале нужно ввести путь до файла (имя) или exit для выхода из программы. В случае некорректного ввода выбор файла предлагается снова. После этого начинается обработка файла и выводятся дублирующиеся записи, статистика этажности домов по городам и затраченное время. По завершении обработки предлагается открыть следующий файл.

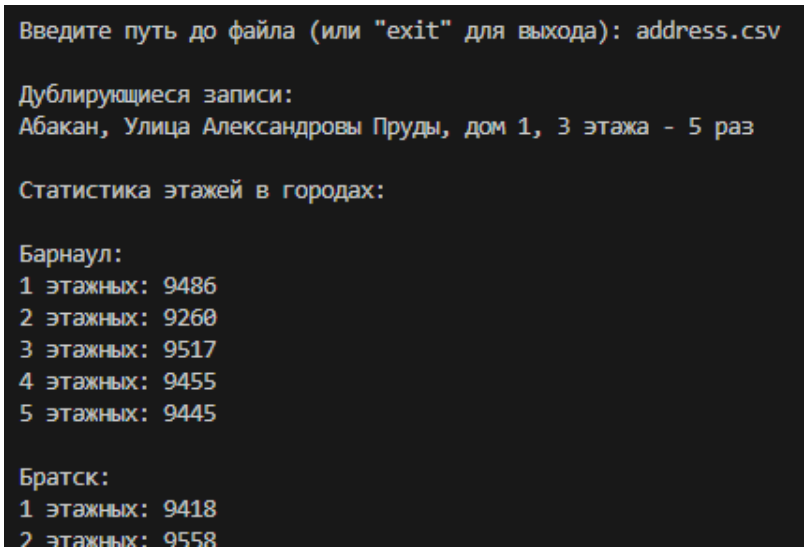

Рисунок 2

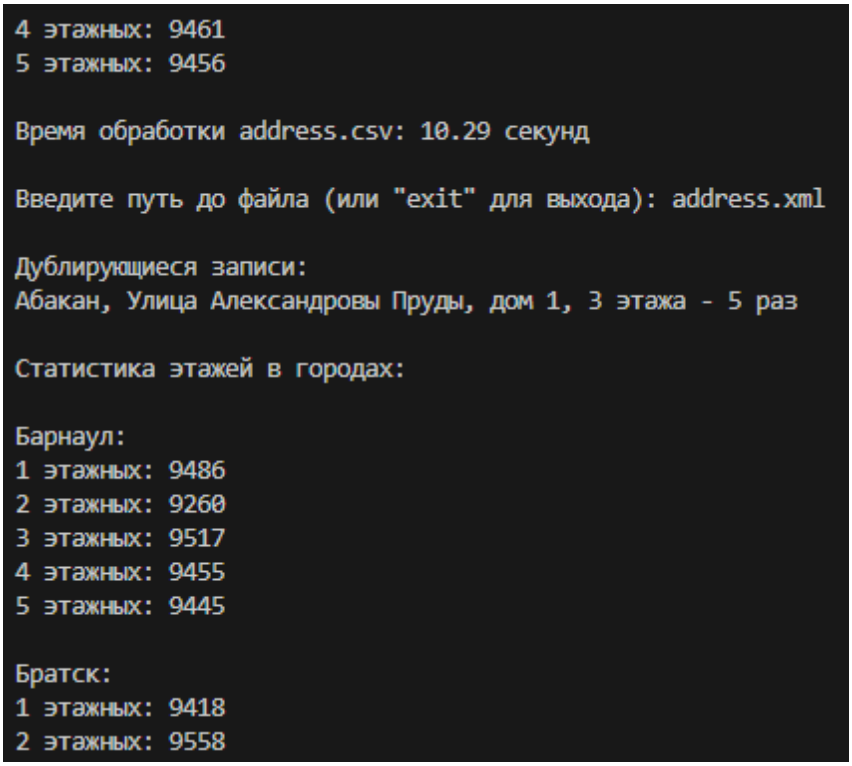

Рисунок 3

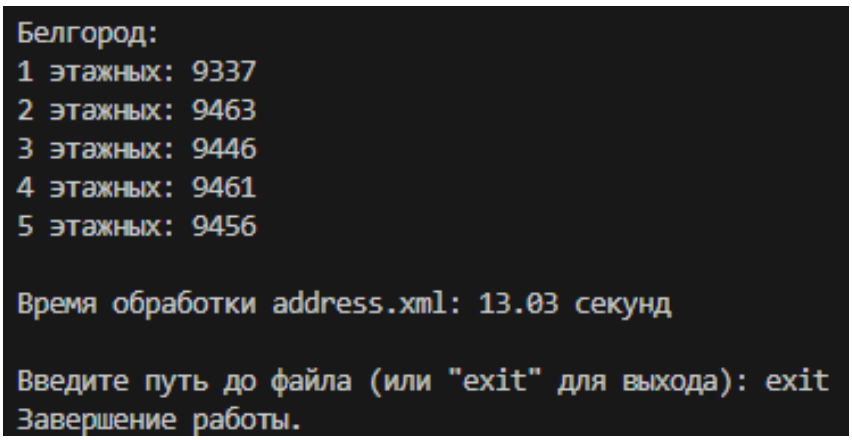

Рисунок 4

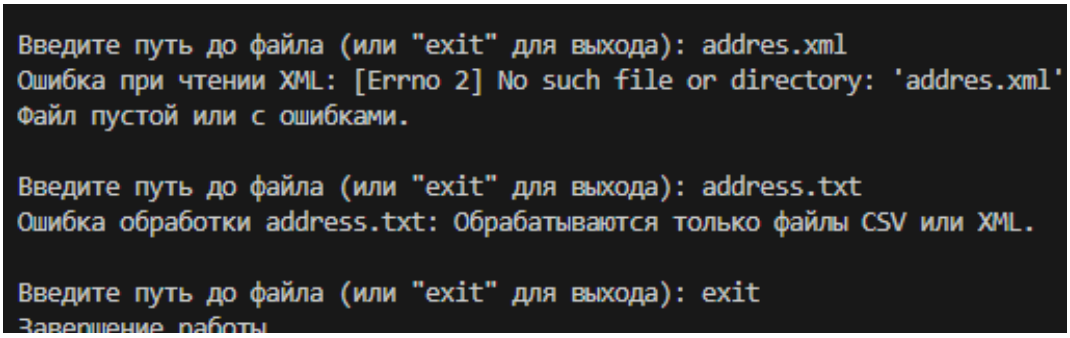

Рисунок 5

Текст программы

«main.py»

```
import time
from parser import FileParsing
from stats import Statistics


class Searcher:
    def run(self):
        while True:
            input_name = input("\nВведите путь до файла (или \"exit\" для выхода): ").strip()
            if input_name.lower() == "exit":
                print("Завершение работы.")
                break
            try:
                start_time = time.time()
                parser = FileParsing(input_name)
                addresses = parser.parse()
                if not addresses:
                    print("Файл пустой или с ошибками.")
                    continue
                statistics = Statistics(addresses)
                duplicates = statistics.find_duplicates()
                floor_stats = statistics.floor_statistics()
                statistics.print_search_results(duplicates, floor_stats)
                elapsed_time = time.time() - start_time
```

```
            print(f"\nВремя обработки {input_name}: {elapsed_time:.2f} секунд")
        except Exception as e:
            print(f"Ошибка обработки {input_name}: {e}")
if __name__ == "__main__":
    app = Searcher()
    app.run()
```

«parser.py»

```
import csv
import xml.etree.ElementTree
from address import Address
class FileParsing:


    def __init__(self, file_path):
        self.file_path = file_path
        self.addresses = []
    def parse(self):
        #Определение способа парсинга
        if self.file_path.endswith('.csv'):
            return self.parse_csv()
        elif self.file_path.endswith('.xml'):
            return self.parse_xml()
        else:
            raise ValueError("Обрабатываются только файлы CSV или XML.")
    def parse_csv(self):
        #Парсинг CSV файла
        try:
```

```
            with open(self.file_path, 'r', encoding='utf-8') as csvfile:
                reader = csv.DictReader(csvfile, delimiter=';')
                for row in reader:
                    self.addresses.append(Address(row["city"], row["street"],
row["house"], row["floor"]))
        except Exception as e:
            print(f"Ошибка при чтении CSV: {e}")
        return self.addresses
    def parse_xml(self):
        #Парсинг XML файла
        try:
            tree = xml.etree.ElementTree.parse(self.file_path)
            root = tree.getroot()
            for item in root.findall("item"):
                self.addresses.append(Address(item.get("city"), item.get("street"),
item.get("house"), item.get("floor")))
        except Exception as e:
            print(f"Ошибка при чтении XML: {e}")
        return self.addresses
}
```

«address.py»

```
class Address:
    def __init__(self, city, street, house, floor):
        self.city = city
        self.street = street
        self.house = int(house)
        self.floor = int(floor)
```

```
    def object(self):
        return (self.city, self.street, self.house, self.floor)
```

«stats.py»

```
from collections import defaultdict
class Statistics:
    def __init__(self, addresses):
        self.addresses = addresses
    def find_duplicates(self):
        #Поиск количества дубликатов
        counter = defaultdict(int)
        for address in self.addresses:
            counter[address.object()] += 1
        duplicates = {key: count for key, count in counter.items() if count > 1}
        return duplicates
    def floor_statistics(self):
        #Статистика по этажам
        stats = defaultdict(lambda: defaultdict(int))
        for address in self.addresses:
            stats[address.city][address.floor] += 1
        return stats
    def print_search_results(self, duplicates, floor_stats):
        #Вывод дубликатов и этажей
        print("\nДублирующиеся записи:")
        for (city, street, house, floor), count in duplicates.items():
            print(f"{city}, {street}, дом {house}, {floor} этажа - {count} раз")
        print("\nСтатистика этажей в городах:")
```

```
for city, floors in floor_stats.items():
    print(f"\n{city}:")
    for floor, count in sorted(floors.items(), key=lambda x: x[0]):
        print(f"{floor} этажных: {count}")
```

Выводы

В результате выполнения работы была разработана программа на Python, считывающая по выбору пользователя csv или xml файлы с адресами для обработки. Программа учитывает некорректный ввод пользователя и ошибки в файлах. Разработанный программный код запускался с помощью python.exe в среде Visual Studio Code. Результаты были выложены на Github:

https://github.com/ChursStudent/OOPprograms