

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

Практическая работа №2
по дисциплине
«Объектно-ориентированное программирование»

Студентка гр. 4351

Байрам Э.

Преподаватель

Кулагин М.В.

Санкт-Петербург
2025

1. Задание

Требуется разработать консольное приложение на Java, которое:

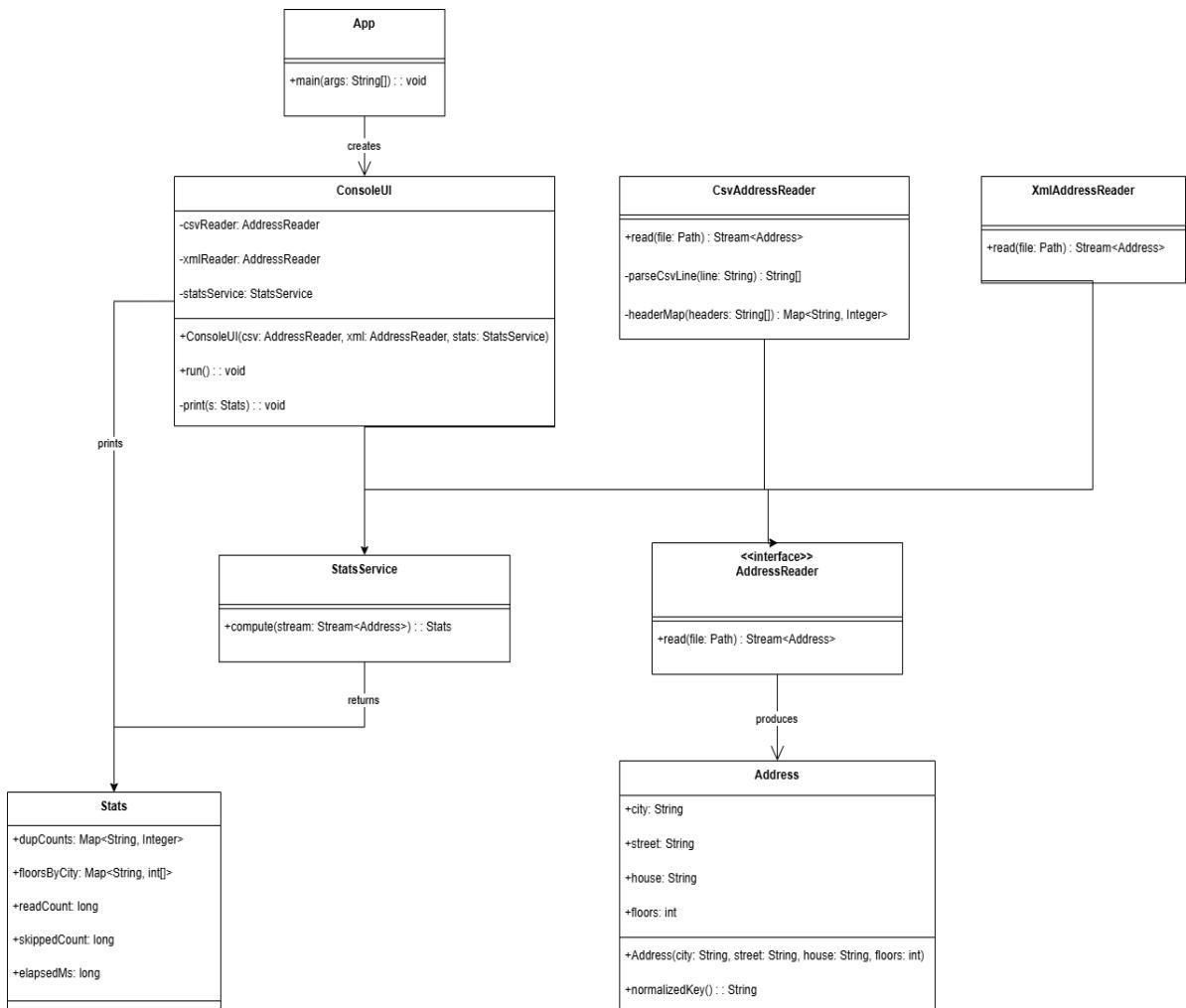
- считывает из консоли **путь к входному файлу** либо команду `q` для завершения работы;
- поддерживает форматы входных данных **CSV** и **XML**;
- читает записи об адресах с полями: **город** (*city*), **улица** (*street*), **дом** (*house*), **этажность** (*floors*, целое число);
- выполняет нормализацию строковых полей (обрезка лишних пробелов, приведение к нижнему регистру, схлопывание повторяющихся пробелов); допускаются синонимы заголовков на разных языках (напр.: *город/sehir/city*, *улица/cadde/street*, *дом/bina/house*, *этажей/kat/floors*);
- формирует агрегированную статистику:
 - **подсчёт дублей адресов** по ключу (*city*, *street*, *house*, *floors*) и вывод записей, встречающихся не реже **двух** раз;
 - **распределение по этажности (1–5 этажей) для каждого города**;
- выводит на экран результаты агрегации, а также служебную информацию: **количество обработанных/пропущенных строк** и **время обработки**;
- работает в цикле до ввода `q`.

Приложение должно корректно обрабатывать **пустой ввод**, **отсутствие файла**, **неподдерживаемое расширение**, а также **ошибки ввода-вывода** и **парсинга** (с краткими сообщениями для пользователя). Для кодировок используется **UTF-8**; при запуске в Windows консоль переводится в `chcp 65001`, а JVM — с параметром `-Dfile.encoding=UTF-8`.

2. Спецификация программы

Диаграмма отражает слоистую архитектуру:

UI (ввод/вывод) → парсеры данных (CSV/XML) → доменная модель (Address) → сервис агрегации (StatsService) → результат (Stats).



Роли классов

- App
Точка входа. Вызывает new ConsoleUI(...).run(). Бизнес-логики не содержит.
- ConsoleUI
Консольный цикл: запрашивает путь к файлу или q, по расширению выбирает ридер, запускает агрегацию и печатает отчёт. Обрабатывает пользовательские ошибки (нет файла, неподдерживаемый формат, пустой ввод).
Зависимости:
 - AddressReader (интерфейс) — выбор конкретной реализации;
 - StatsService — вычисление статистики;
 - печатает экземпляр Stats.
- AddressReader (интерфейс)
Единый контракт чтения: read(file: Path) → Stream<Address>.
Гарантии: нормализация строк (trim, lowerCase, схлопывание пробелов), пропуск некорректных записей без падения приложения.
- CsvAddressReader (реализация AddressReader)
Построчно парсит CSV (учитывает кавычки/экраннирование), сопоставляет заголовки с синонимами (город/sehir/city и т. п.), порождает поток Address.
- XmlAddressReader (реализация AddressReader)
SAX-разбор XML; поддержка синонимов тегов; на выходе поток Address.
- Address
Простая доменная сущность: city, street, house, floors. Содержит метод формирования нормализованного ключа (city|street|house|floors) для поиска дублей.
- StatsService
Принимает Stream<Address>, вычисляет:
 - карту дублей dupCounts: Map<String, Integer> по ключу адреса (выводятся записи с count ≥ 2);
 - распределение домов по этажности для каждого города floorsByCity: Map<String, int[1..5]>;
 - счётчики readCount / skippedCount и elapsedMs.
Возвращает агрегат Stats.
- Stats
Переносчик результата: структуры, описанные выше, плюс служебные метрики.

Связи на диаграмме

- App → ConsoleUI — создаёт экземпляр UI.
- ConsoleUI → AddressReader — использует (выбор CSV/XML по расширению).
- CsvAddressReader ..|> AddressReader, XmlAddressReader ..|> AddressReader — реализация интерфейса.

- AddressReader → Address — порождает доменные объекты.
- ConsoleUI → StatsService — использует сервис агрегации.
- StatsService → Stats — возвращает агрегированный результат.
- ConsoleUI → Stats — печатает пользователю.

Сценарий работы (сквозной поток)

1. App.main создаёт ConsoleUI и запускает run().
2. Пользователь вводит путь; UI выбирает CsvAddressReader или XmlAddressReader.
3. Ридер читает файл → Stream<Address>.
4. StatsService.compute(stream) считает дубли и распределение по этажности → Stats.
5. UI форматирует и выводит блоки:
 - **Duplicates (count ≥ 2)**
 - **Floors per city (1F...5F)**
 - **Processed / skipped / Processing time**
 Затем цикл повторяется до q.

Принципы проектирования

- SRP: каждая сущность отвечает за свою зону (UI, парсинг, модель, агрегация).
- OCP: легко добавить новый формат (напр., JsonAddressReader) без изменения ConsoleUI — достаточно одной реализации AddressReader.
- Устойчивость к ошибкам: некорректные строки пропускаются, пользователю выводятся краткие сообщения.

3. Описание интерфейса пользователя программы

Ниже показано, как выглядит взаимодействие пользователя с приложением в консоли.

Общий диалог

Приложение работает в цикле. Каждую итерацию оно выводит приглашение:

```
C:\Users\Esra Bayram\OneDrive\Masaüstü\odev2_plain>compile.bat  
Compiled to .\out  
  
C:\Users\Esra Bayram\OneDrive\Masaüstü\odev2_plain>run.bat  
File path (or q to quit): -
```

Пользователь вводит путь к файлу **CSV** или **XML** (напр., docs\sample.csv или docs\sample.xml) и получает агрегированный отчёт. Для выхода — ввести **q** и нажать Enter.

Пример 1 — успешная обработка CSV

Ввод:

```
C:\Users\Esra Bayram\OneDrive\Masaüstü\odev2_plain>run.bat  
File path (or q to quit): docs\sample.csv  
  
Duplicates (count >= 2):  
    kazan|lenina|10|5 -> 2
```

Вывод:

```
Duplicates (count >= 2):  
    kazan|lenina|10|5 -> 2  
  
Floors per city:  
    Naberezhnye : 1F=0      2F=0      3F=1      4F=0      5F=0  
    Kazan      : 1F=1      2F=0      3F=0      4F=0      5F=2  
  
Processed: 4 rows, skipped: 0  
Processing time: 22 ms  
  
File path (or q to quit):
```

Пояснение:

- Блок Duplicates показывает адреса, встретившиеся не реже 2 раз (ключ: city|street|house|floors).
- Блок Floors per city — распределение домов по этажности (1–5 этажей) для каждого города.
- Далее печатаются служебные метрики: обработано/пропущено строк и затраченное время.

Пример 2 — успешная обработка XML

Ввод:

```
C:\Users\Esra Bayram\OneDrive\Masaüstü\odev2_plain>run.bat
:File path (or q to quit): docs\sample.xml
```

Вывод по структуре аналогичен Примеру 1 (дубли, распределение по этажам, метрики).

Поддерживаются теги-синонимы на разных языках (например, city/sehir/город, street/cadde/улица, house/bina/дом, floors/kat/этажей).

Обработка ошибок и некорректного ввода

- Несуществующий файл

```
C:\Users\Esra Bayram\OneDrive\Masaüstü\odev2_plain>run.bat
:File path (or q to quit): docs\sample.xml
:File not found.
:File path (or q to quit):
```

- Неподдерживаемое расширение (например, .txt) — краткое сообщение и повтор приглашения.
- Пустой ввод — повтор приглашения без падения приложения.
- Повреждённые/неполные строки в файле — строка пропускается, счётчик skipped увеличивается; приложение продолжает работу.
- Ошибки ввода-вывода / парсинга — печатается краткое сообщение (I/O / parse error), затем цикл продолжается.

Кодировка и запуск в Windows

Для корректного отображения кириллицы/турецких символов используется UTF-8:

- перед запуском консоли: chcp 65001;
- для JVM: параметр -Dfile.encoding=UTF-8 (уже включён в скрипт run.bat).

4. Текст программы

App.java

```
import java.nio.charset.StandardCharsets;

public class App {

    public static void main(String[] args) {

        ConsoleUI ui = new ConsoleUI(
            new CsvAddressReader(),
            new XmlAddressReader(),
            new StatsService());

        ui.run();
    }
}
```

ConsoleUI . java

```
import java.io.IOException;
import java.nio.file.*;
import java.util.Locale;
import java.util.Scanner;
```

```
public class ConsoleUI {
```

```
private final AddressReader csvReader;
private final AddressReader xmlReader;
private final StatsService statsService;

public ConsoleUI(AddressReader csvReader, AddressReader
xmlReader, StatsService statsService) {
    this.csvReader = csvReader;
    this.xmlReader = xmlReader;
    this.statsService = statsService;
}

public void run() {
    Locale.setDefault(Locale.ROOT);
    try (Scanner sc = new Scanner(System.in,
java.nio.charset.StandardCharsets.UTF_8)) {
        while (true) {
            System.out.print("File path (or q to quit): ");
            String line = sc.nextLine().trim();
            if (line.equalsIgnoreCase("q")) break;
            if (line.isEmpty()) continue;
            Path file = Paths.get(line.replace("\\\"", ""));
            if (!Files.exists(file)) {
                System.out.println("File not found.");
                continue;
            }
        }
    }
}
```

```
}

    String name =
file.getFileName().toString().toLowerCase(Locale.ROOT);

    try {

        Stats stats;

        if (name.endsWith(".csv")) {

            stats = statsService.compute(csvReader.read(file));

        } else if (name.endsWith(".xml")) {

            stats = statsService.compute(xmlReader.read(file));

        } else {

            System.out.println("Unsupported file type (use .csv or
.xml).");

            continue;

        }

        print(stats);

    } catch (IOException ex) {

        System.out.println("I/O error: " + ex.getMessage());

    } catch (RuntimeException ex) {

        System.out.println("Parse error: " + ex.getMessage());

    }

}

}
```

```
private void print(Stats s) {  
    System.out.println();  
    System.out.println("Duplicates (count >= 2):");  
    s.dupCounts.entrySet().stream()  
        .filter(e -> e.getValue() >= 2)  
        .forEach(e -> System.out.println(" " + e.getKey() + " -> " +  
            e.getValue()));  
    if (s.dupCounts.values().stream().noneMatch(v -> v >= 2)) {  
        System.out.println(" (none)"); }  
    System.out.println();  
    System.out.println("Floors per city:");  
    s.floorsByCity.forEach((city, buckets) -> {  
        System.out.printf(" %-11s: 1F=%d 2F=%d 3F=%d 4F=%d  
5F=%d%n",  
            city, buckets[1], buckets[2], buckets[3], buckets[4],  
            buckets[5]);  
    });  
    System.out.println();  
    System.out.printf("Processed: %d rows, skipped: %d%n",  
        s.readCount, s.skippedCount);  
    System.out.printf("Processing time: %d ms%n", s.elapsedMs);  
    System.out.println();  
}  
}
```

AddressReader.java

```
import java.io.IOException;
import java.nio.file.Path;
import java.util.stream.Stream;

public interface AddressReader {
    Stream<Address> read(Path file) throws IOException;
}
```

Address . java

```
import java.util.Locale;
public class Address {
    public final String city;
    public final String street;
    public final String house;
    public final int floors;

    public Address(String city, String street, String house, int floors) {
        this.city = normalize(city);
        this.street = normalize(street);
        this.house = normalize(house);
        this.floors = floors;
    }
}
```

```
public String key() {  
    return city + "|" + street + "|" + house + "|" + floors;  
}
```

```
private static String normalize(String s) {  
    if (s == null) return "";  
    String t = s.trim().toLowerCase(Locale.ROOT);  
    return t.replaceAll("\\s+", " ");  
}  
}
```

CsvAddressReader.java

```
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.*;  
import java.util.*;  
import java.util.stream.Stream;
```

```
public class CsvAddressReader implements AddressReader {
```

```
private static final Map<String, String> ALIASES = Map.ofEntries(  
    Map.entry("city", "city"), Map.entry("sehir", "city"),  
    Map.entry("şehir", "city"),  
    Map.entry("город", "city"), Map.entry("gorod", "city"),  
    Map.entry("street", "street"), Map.entry("cadde", "street"),  
    Map.entry("улица", "street"),  
    Map.entry("house", "house"), Map.entry("bina", "house"),  
    Map.entry("дом", "house"),  
    Map.entry("floors", "floors"), Map.entry("kat", "floors"),  
    Map.entry("этажей", "floors")  
);  
  
@Override  
  
public Stream<Address> read(Path file) throws IOException {  
    List<String> lines = Files.readAllLines(file,  
    StandardCharsets.UTF_8);  
    if (lines.isEmpty()) return Stream.empty();  
  
    String[] header = parse(lines.get(0));  
    Map<String, Integer> idx = headerMap(header);  
  
    List<Address> out = new ArrayList<>();  
    for (int i = 1; i < lines.size(); i++) {  
        String[] cells = parse(lines.get(i));  
        try {  
            String city = cell(cells, idx.get("city"));  
            Address address = new Address(city, null, null, null, null);  
            out.add(address);  
        } catch (Exception e) {  
            System.out.println("Error reading line " + i + ": " + e.getMessage());  
        }  
    }  
    return out.stream();  
}
```

```
        String street = cell(cells, idx.get("street"));

        String house = cell(cells, idx.get("house"));

        int floors = Integer.parseInt(cell(cells, idx.get("floors")));

        out.add(new Address(city, street, house, floors));

    } catch (Exception ignore) {

        // пропускаем «грязные» строки

    }

}

return out.stream();

}

private static Map<String, Integer> headerMap(String[] header) {

    Map<String, Integer> map = new HashMap<>();

    for (int i = 0; i < header.length; i++) {

        String raw = header[i].trim().toLowerCase(Locale.ROOT);

        String canonical = ALIASES.get(raw);

        if (canonical != null && !map.containsKey(canonical))
            map.put(canonical, i);

        if (!map.keySet().containsAll(List.of("city", "street", "house",
                "floors")))

            throw new IllegalArgumentException("Header must contain
                city/street/house/floors (with aliases).");

    }

    return map;

}
```

```
private static String cell(String[] row, Integer i) { return (i == null || i >= row.length) ? "" : row[i]; }
```

```
private static String[] parse(String line) {  
    List<String> res = new ArrayList<>();  
    StringBuilder sb = new StringBuilder();  
    boolean q = false;  
    for (int i = 0; i < line.length(); i++) {  
        char c = line.charAt(i);  
        if (c == '"') {  
            if (q && i + 1 < line.length() && line.charAt(i + 1) == '"') {  
                sb.append(""); i++; // экранированная кавычка  
            } else {  
                q = !q;  
            }  
        } else if (c == ',' && !q) {  
            res.add(sb.toString()); sb.setLength(0);  
        } else {  
            sb.append(c);  
        }  
    }  
    res.add(sb.toString());
```

```
        return res.toArray(new String[0]);  
    }  
}
```

XmlAddressReader.java

```
import org.xml.sax.*;  
import org.xml.sax.helpers.DefaultHandler;  
  
import javax.xml.parsers.SAXParser;  
import javax.xml.parsers.SAXParserFactory;  
import java.io.IOException;  
import java.nio.file.Path;  
import java.util.*;  
import java.util.stream.Stream;  
  
public class XmlAddressReader implements AddressReader {  
  
    private static final Map<String, String> ALIASES = Map.ofEntries(  
        Map.entry("city", "city"), Map.entry("sehir", "city"),  
        Map.entry("şehir", "city"),  
        Map.entry("город", "city"), Map.entry("gorod", "city"),  
        Map.entry("street", "street"), Map.entry("cadde", "street"),  
        Map.entry("улица", "street"),  
        Map.entry("house", "house"), Map.entry("bina", "house"),  
        Map.entry("дом", "house"),
```

```
        Map.entry("floors", "floors"), Map.entry("kat", "floors"),
Map.entry("этажей", "floors")
);
@Override
public Stream<Address> read(Path file) throws IOException {
    try {
        SAXParserFactory f = SAXParserFactory.newInstance();
        SAXParser p = f.newSAXParser();
        List<Address> out = new ArrayList<>();
        p.parse(file.toFile(), new Handler(out));
        return out.stream();
    } catch (Exception e) {
        throw new IOException("XML parse error: " + e.getMessage(),
e);
    }
}
```

```
private static class Handler extends DefaultHandler {
    private final List<Address> out;
    private final Map<String, String> fields = new HashMap<>();
    private final StringBuilder buf = new StringBuilder();
    private boolean inRecord = false;

    Handler(List<Address> out) { this.out = out; }

    @Override public void startElement(String uri, String localName,
Attributes atts) {
    buf.setLength(0);
```

```

String tag = canonical(qName);
if ("record".equalsIgnoreCase(tag)) ||

"addr".equalsIgnoreCase(tag)) inRecord = true;
}

@Override public void characters(char[] ch, int start, int length) {
buf.append(ch, start, length); }

@Override public void endElement(String uri, String local, String
qName) {

String tag = canonical(qName);
if (inRecord && ALIASES.containsKey(tag)) {
    fields.put(ALIASES.get(tag), buf.toString());
}
if ("record".equalsIgnoreCase(tag)) ||

"addr".equalsIgnoreCase(tag)) {

try {
    Address a = new Address(
        fields.getDefault("city", ""),
        fields.getDefault("street", ""),
        fields.getDefault("house", ""),
        Integer.parseInt(fields.getDefault("floors", "0")))
    );
    out.add(a);
} catch (Exception ignore) { /* skip bad items */ }
fields.clear();
inRecord = false;
}
}

```

```
    private static String canonical(String name) {  
        return name.trim().toLowerCase(Locale.ROOT);  
    }  
}  
}
```

Stats.java

```
import java.util.LinkedHashMap;  
import java.util.Map;  
import java.util.TreeMap;  
  
public class Stats {  
    public final Map<String, Integer> dupCounts = new  
LinkedHashMap<>();  
    public final Map<String, int[]> floorsByCity = new TreeMap<>();  
    public long readCount;  
    public long skippedCount;  
    public long elapsedMs;  
}
```

StatsService.java

```
import java.util.Map;
import java.util.concurrent.atomic.AtomicLong;
import java.util.function.Consumer;
import java.util.stream.Stream;

public class StatsService {

    public Stats compute(Stream<Address> stream) {
        Stats s = new Stats();
        long t0 = System.nanoTime();

        AtomicLong read = new AtomicLong();
        AtomicLong skipped = new AtomicLong();

        Consumer<Address> consume = (Address a) -> {
            if (a == null || a.floors < 1 || a.floors > 5) {
                skipped.incrementAndGet();
                return;
            }
            read.incrementAndGet();
        };

        // Дубли
        s.dupCounts.merge(a.key(), 1, Integer::sum);

        // Распределение этажности по городам
```

```
    s.floorsByCity.computeIfAbsent(a.city,      k      ->      new
int[6])[a.floors]++;
}

stream.forEach(consume);

s.readCount = read.get();
s.skippedCount = skipped.get();
s.elapsedMs  =  Math.max(0, (System.nanoTime() - t0) /
1_000_000);
return s;
}
}
```

5. Выводы

Итог. Реализовано консольное приложение на Java 17, которое читает адресные данные из CSV и XML, нормализует поля, считает дубли по ключу (city|street|house|floors) и строит *распределение по этажности (1–5 этажей) для каждого города. Программа работает циклически до команды q, устойчиво обрабатывает ошибки ввода-вывода и «грязные» строки, выводит служебные метрики (обработано/пропущено, время).

Соответствие заданию. Все пункты ТЗ выполнены:

- поддержка двух форматов (CSV, XML) через интерфейс AddressReader и две реализации;
- нормализация строковых полей и поддержка многоязычных синонимов заголовков/тегов (ru/tr/en);
- подсчёт дублей и агрегирование по этажности; печать отчёта в консоль;
- корректная обработка пустого ввода, отсутствия файла, неподдерживаемого расширения, I/O и parse-ошибок;
- кодировка **UTF-8** (консоль chcp 65001, JVM -Dfile.encoding=UTF-8).

Качество и архитектура. Использована слойстая схема: UI → ридеры → доменная модель → сервис агрегации. Принципы **SRP/OCP** соблюdenы: формат легко расширить (добавить `JsonAddressReader`), UI и расчётная логика развязаны.