Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Звіт

з лабораторної роботи No 5
з дисципліни

«Мова програмування Java»

Виконала студентка групи ЗПІ_зп-41
Павлюк Р.В.

Перевірив Орленко С.П.

Київ 2025

У цій лабораторній роботі я виконала три завдання, спрямовані на вивчення різних типів потоків введення-виведення в Java, роботу з фільтрами та мережевими ресурсами.

Виконання завдання 1

Виконання завдання 1 представлено в файлі MaxWordsLine.java - реалізовано алгоритм пошуку рядків із максимальною кількістю слів у текстовому файлі.

Програма реалізує алгоритм аналізу текстового файлу через потокове читання за допомогою BufferedReader, що забезпечує стабільну роботу з великими обсягами даних без надмірного навантаження на оперативну пам'ять. Процес обробки відбувається порядково: кожен рядок зчитується та аналізується окремо.

Для точного підрахунку слів застосовується регулярний вираз \\s+. Це гарантує, що програма враховує лише реальні слова, ігноруючи будь-яку кількість роздільників між ними, включаючи множинні пробіли та символи табуляції.

Алгоритм передбачає обробку випадків, коли у файлі міститься кілька рядків з однаковою рекордною кількістю слів. Для цього використовується динамічний список, який повністю оновлюється при знаходженні нового, довшого рядка, або доповнюється, якщо кількість слів у поточному рядку збігається з уже знайденим максимумом. Це дозволяє отримати повний перелік усіх найбільш інформативних рядків наприкінці роботи програми.

Нижче представлений код програми

task_5_1 > MaxWordsLine.java

```java
import java.io.*;
import java.util.*;

public class MaxWordsLine {
    public static void main(String[] args) {
        String fileName = "input.txt";
        List<String> maxLines = new ArrayList<>();
        int maxCount = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String currentLine;
            while ((currentLine = br.readLine()) != null) {
                // Використовуємо регулярний вираз \\s+, щоб врахувати будь-яку кількість пробілів
                String[] words = currentLine.trim().split("\\s+");

                // Перевірка на порожній рядок
                int currentCount = (currentLine.trim().isEmpty()) ? 0 : words.length;

                if (currentCount > maxCount) {
                    maxCount = currentCount;
                    maxLines.clear(); // Знайдено новий рекорд, очищуємо старі результати
                    maxLines.add(currentLine);
                } else if (currentCount == maxCount && maxCount > 0) {
                    maxLines.add(currentLine); // Рядок з такою ж кількістю слів
                }
            }
```

```
26              }
27
28              if (maxLines.isEmpty()) {
29                  System.out.println("Файл порожній або не містить слів.");
30              } else {
31                  System.out.println("Максимальна кількість слів: " + maxCount);
32                  System.out.println("Знайдено рядків: " + maxLines.size());
33                  System.out.println("--- Список рядків ---");
34                  for (String line : maxLines) {
35                      System.out.println(line);
36                  }
37              }
38          } catch (IOException e) {
39              System.err.println("Помилка читання файлу: " + e.getMessage());
40          }
41      }
42  }
```

Результати компіляції та демонстрація роботи програми наведені нижче. Перший скріншот демонструє роботу програми для пустого файлу input.txt

```
ruszlanapavliuk@POLO2-0349 task_5_1 % java MaxWordsLine.java
Файл порожній або не містить слів.
```

Перевіряю для сценарію, в якому файл містить декілька рядків з однаковою кількістю слів

```
task_5_1 > input.txt
1    so we commissioned a document
2    about sustenance and the city's pores
3    about sustenance and the city's pores
4    metaphors of food and skin
5    for when the water rises
```

```
ruszlanapavliuk@POLO2-0349 task_5_1 % java MaxWordsLine.java
Максимальна кількість слів: 6
Знайдено рядків: 2
--- Список рядків ---
about sustenance and the city's pores
about sustenance and the city's pores
```

Перевіряю для сценарію, де є лише один рядок з найбільшою кількістю слів

task_5_1 > input.txt

```
1   so we commissioned a document
2   about sustenance and the city's pores
3   metaphors of food and skin
4   for when the water rises
```

```
about sustenance and the city's pores
ruszlanapavliuk@POLO2-0349 task_5_1 % java MaxWordsLine.java
Максимальна кількість слів: 6
Знайдено рядків: 1
--- Список рядків ---
about sustenance and the city's pores
ruszlanapavliuk@POLO2-0349 task_5_1 %
```

Виконання завдання 2

Виконання завдання 2 представлено в файлі CaesarCipher.java.

Логіка захисту даних базується на механізмі фільтрації потоків через використання класів FilterWriter та FilterReader. Такий підхід дозволяє інтегрувати процес перетворення символів безпосередньо в операції запису та читання.

Процес шифрування реалізовано шляхом перевизначення методу write() у класі-обгортці. Кожен символ тексту зміщується на значення коду ключового символу перед тим, як потрапити у файл. Дешифрування працює за дзеркальним принципом: під час читання файлу через метод read() значення ключа віднімається від коду кожного символу, повертаючи текст до початкового вигляду. Використання цих типів потоків дозволяє відокремити логіку обробки даних від логіки роботи з файловою системою.

Нижче представлений код програми

task_5_2 > CaesarCipher.java

```java
import java.io.*;

public class CaesarCipher {

    // Власний фільтр для шифрування
    static class CaesarWriter extends FilterWriter {
        private final char key;
        protected CaesarWriter(Writer out, char key) { super(out); this.key = key; }

        @Override
        public void write(int c) throws IOException {
            super.write(c + key); // Шифрування: зміщення коду символу вгору
        }
    }
```

```java
    // Власний фільтр для дешифрування
    static class CaesarReader extends FilterReader {
        private final char key;
        protected CaesarReader(Reader in, char key) { super(in); this.key = key; }

        @Override
        public int read() throws IOException {
            int c = super.read();
            return (c == -1) ? -1 : (c - key); // Дешифрування: зміщення коду вниз
        }
    }

    public static void main(String[] args) throws IOException {
        char key = 'B'; // Ключовий символ
        String originalText = "Hello Java Streams!";
        String fileName = "encrypted.txt";

        // a) Метод шифрування через FilterWriter
        try (CaesarWriter cw = new CaesarWriter(new FileWriter(fileName), key)) {
            cw.write(originalText);
        }

        // b) Метод дешифрування через FilterReader
        StringBuilder decrypted = new StringBuilder();
        try (CaesarReader cr = new CaesarReader(new FileReader(fileName), key)) {
            int data;
            while ((data = cr.read()) != -1) {
                decrypted.append((char) data);
            }
        }

        System.out.println("Оригінал: " + originalText);
        System.out.println("Відновлено: " + decrypted);
    }
}
```

Результати компіляції та демонстрація роботи програми наведені нижче

```
ruszlanapavliuk@POLO2-0349 task_5_2 % java CaesarCipher.java
Оригінал: Hello Java Streams!
Відновлено: #**-4□20#+1□
```

task_5_2 > encrypted.txt

```
1   Hello Java Streams!
```

Виконання завдання 3

Виконання завдання 3 представлено в файлі TagFrequency.java. - аналіз частоти тегів за URL-адресою.

Робота програми починається з встановлення мережевого з'єднання через URL.openStream(), що дозволяє отримувати HTML-код сторінки у

вигляді вхідного потоку даних. Для обробки отриманого тексту використовується обгортка над InputStreamReader, яка перетворює байти в символи для подальшого аналізу.

Виявлення тегів здійснюється за допомогою регулярних виразів, які ідентифікують відкриваючі теги та виокремлюють їхні назви. Отримані дані накопичуються у структурі Map, де ключем є назва тегу, а значенням — частота його появи. Фінальна частина логіки передбачає подвійне сортування: спочатку за алфавітом (лексикографічно), а потім за частотою використання (від найрідших до найпоширеніших), що забезпечує комплексний аналіз структури веб-сторінки.

Нижче представлений код

task_5_3 > TagFrequency.java

```java
import java.io.*;
import java.net.URL;
import java.util.*;
import java.util.regex.*;

public class TagFrequency {
    public static void main(String[] args) throws IOException {
        String urlString = "https://www.google.com"; // URL на вибір
        URL url = new URL(urlString);
        Map<String, Integer> tagMap = new HashMap<>();

        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String inputLine;
            // Регулярний вираз для пошуку назв тегів (напр. <div, <p)
            Pattern pattern = Pattern.compile("<([a-zA-Z1-6]+)");

            while ((inputLine = in.readLine()) != null) {
                Matcher matcher = pattern.matcher(inputLine);
                while (matcher.find()) {
                    String tag = matcher.group(1).toLowerCase();
                    tagMap.put(tag, tagMap.getOrDefault(tag, 0) + 1);
                }
            }
        }

        // a) Сортування в лексикографічному порядку (за назвою)
        System.out.println("--- Лексикографічний порядок ---");
        tagMap.entrySet().stream()
                .sorted(Map.Entry.comparingByKey())
                .forEach(e -> System.out.println(e.getKey() + ": " + e.getValue()));

        // b) Сортування за частотою появи
        System.out.println("\n--- За частотою (зростання) ---");
        tagMap.entrySet().stream()
                .sorted(Map.Entry.comparingByValue())
                .forEach(e -> System.out.println(e.getKey() + ": " + e.getValue()));
    }
}
```

Результати компіляції та демонстрація роботи програми наведені нижче

```
ruszlanapavliuk@POLO2-0349 task_5_3 % java TagFrequency.java
--- Лексикографічний порядок ---
a: 18
b: 1
body: 1
br: 5
c: 1
center: 1
div: 11
form: 1
head: 1
html: 1
img: 2
input: 10
meta: 2
nobr: 2
p: 1
script: 7
span: 8
style: 2
table: 1
td: 3
title: 1
tr: 1
u: 1
```

```
--- За частотою (зростання) ---
b: 1
c: 1
center: 1
title: 1
body: 1
head: 1
p: 1
form: 1
u: 1
html: 1
table: 1
tr: 1
nobr: 2
img: 2
meta: 2
style: 2
td: 3
br: 5
script: 7
span: 8
input: 10
div: 11
a: 18
```