

## ЛАБОРАТОРНА РОБОТА №3

### Утилітарні класи. Обробка масивів і рядків

Мета:

- розробка власних утилітарних класів;
- набуття навичок вирішення прикладних задач з використанням масивів і рядків.

Вимоги

- розробити та продемонструвати консольну програму мовою *Java* в середовищі *Eclipse* для вирішення прикладної задачі за номером, що відповідає збільшеному на одиницю залишку від ділення на 15 зменшеного на одиницю номера студента в журналі групи;
- при вирішенні прикладних задач використовувати латинку;
- продемонструвати використання об'єктів класу *StringBuilder* або *StringBuffer*.
- застосувати функціональну (процедурну) декомпозицію - розробити власні утилітарні класи (особливий випадок допоміжного класу, див. *Helper Class*) та для обробки даних використовувати відповідні статичні методи.
- Забороняється використовувати засоби обробки регулярних виразів: класи пакету *java.util.regex* (*Pattern*, *Matcher* та ін.), а також відповідні методи класу *String* (*matches*, *replace*, *replaceFirst*, *replaceAll*, *split*).

Варіант 13:

Ввести текст. Текст розбити на речення. Для кожного речення знайти та надрукувати всі слова максимальної та всі слова мінімальної довжини.

Результат вивести у вигляді таблиці.

## ЗМІСТ

1. Індивідуальне завдання .....	1
2. Розробка програми .....	3
2.1. Опис програми .....	3
2.2. Важливі фрагменти програми .....	3
3. РЕЗУЛЬТАТИ .....	5
4. ВИСНОВКИ .....	6

## 2. Опис програми

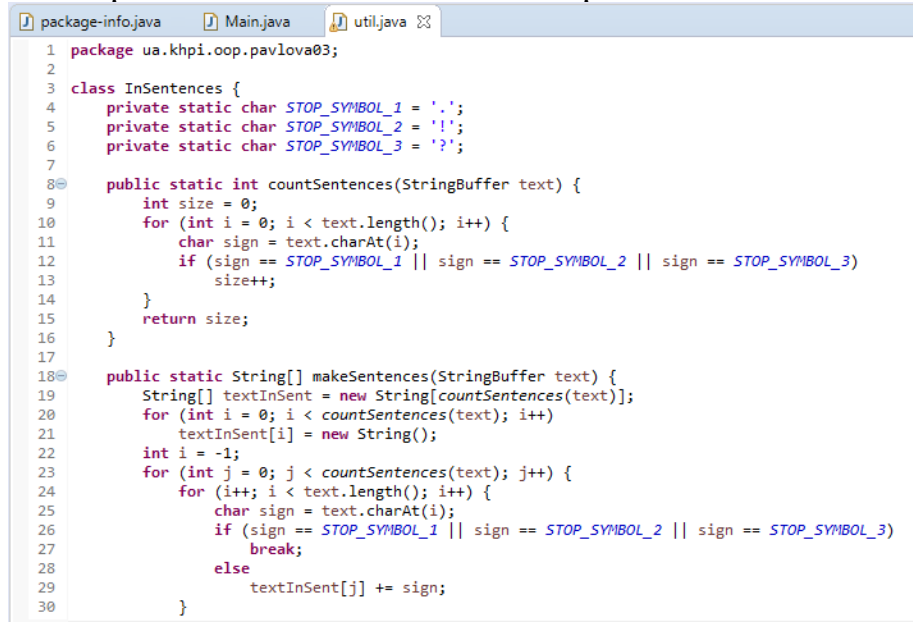
### 2.1. Розробка програми

Для виконання індивідуального завдання за допомогою декомпозиції задачу було розділено на кілька підзадач:

- розділення тексту на речення
- пошук кількості слів у реченні
- пошук максимального/мінімального

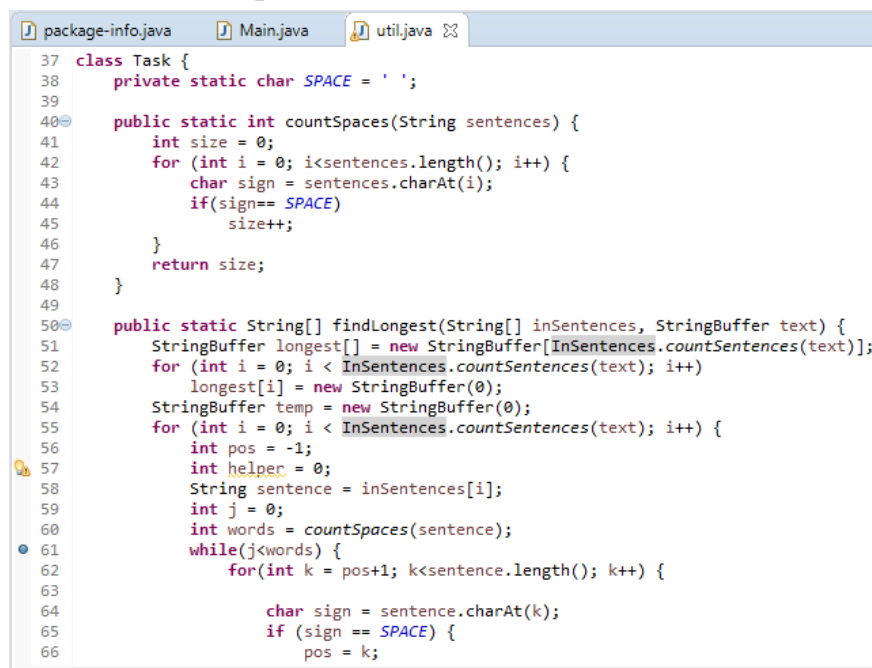
### 2.2. Важливі фрагменти програми

На рис.1-2. можна побачити створені допоміжні класи



```
1 package ua.khpi.oop.pavlova03;
2
3 class InSentences {
4     private static char STOP_SYMBOL_1 = '.';
5     private static char STOP_SYMBOL_2 = '!';
6     private static char STOP_SYMBOL_3 = '?';
7
8     public static int countSentences(StringBuffer text) {
9         int size = 0;
10        for (int i = 0; i < text.length(); i++) {
11            char sign = text.charAt(i);
12            if (sign == STOP_SYMBOL_1 || sign == STOP_SYMBOL_2 || sign == STOP_SYMBOL_3)
13                size++;
14        }
15        return size;
16    }
17
18    public static String[] makeSentences(StringBuffer text) {
19        String[] textInSent = new String[countSentences(text)];
20        for (int i = 0; i < countSentences(text); i++)
21            textInSent[i] = new String();
22        int i = -1;
23        for (int j = 0; j < countSentences(text); j++) {
24            for (i++; i < text.length(); i++) {
25                char sign = text.charAt(i);
26                if (sign == STOP_SYMBOL_1 || sign == STOP_SYMBOL_2 || sign == STOP_SYMBOL_3)
27                    break;
28                else
29                    textInSent[j] += sign;
30            }
31        }
32    }
33 }
```

рис.1. Клас InSentences



```
37 class Task {
38     private static char SPACE = ' ';
39
40     public static int countSpaces(String sentences) {
41         int size = 0;
42         for (int i = 0; i < sentences.length(); i++) {
43             char sign = sentences.charAt(i);
44             if (sign == SPACE)
45                 size++;
46         }
47         return size;
48     }
49
50     public static String[] findLongest(String[] inSentences, StringBuffer text) {
51         StringBuffer longest[] = new StringBuffer[inSentences.countSentences(text)];
52         for (int i = 0; i < inSentences.countSentences(text); i++)
53             longest[i] = new StringBuffer(0);
54         StringBuffer temp = new StringBuffer(0);
55         for (int i = 0; i < inSentences.countSentences(text); i++) {
56             int pos = -1;
57             int helper = 0;
58             String sentence = inSentences[i];
59             int j = 0;
60             int words = countSpaces(sentence);
61             while (j < words) {
62                 for (int k = pos + 1; k < sentence.length(); k++) {
63
64                     char sign = sentence.charAt(k);
65                     if (sign == SPACE) {
66                         pos = k;
```

рис.2. Клас Task

Клас InSentences було створено для підготовки вхідного тексту до виконання індивідуального завдання, тобто він містить функції для рахування речень та переробку тексту у масив строк(речень).

Клас Task було створено для самого виконання індивідуального завдання, тобто було написано функції для пошуку найдовшого та найкоротшого слова.

На рис.3-4. можна побачити функції класу InSentences

```

8- public static int countSentences(StringBuffer text) {
9     int size = 0;
10    for (int i = 0; i < text.length(); i++) {
11        char sign = text.charAt(i);
12        if (sign == STOP_SYMBOL_1 || sign == STOP_SYMBOL_2 || sign == STOP_SYMBOL_3)
13            size++;
14    }
15    return size;
16 }
17

```

рис.3. Функція для рахування кількості речень

```

17
18- public static String[] makeSentences(StringBuffer text) {
19    String[] textInSent = new String[countSentences(text)];
20    for (int i = 0; i < countSentences(text); i++)
21        textInSent[i] = new String();
22    int i = -1;
23    for (int j = 0; j < countSentences(text); j++) {
24        for (i++; i < text.length(); i++) {
25            char sign = text.charAt(i);
26            if (sign == STOP_SYMBOL_1 || sign == STOP_SYMBOL_2 || sign == STOP_SYMBOL_3)
27                break;
28            else
29                textInSent[j] += sign;
30        }
31    }
32    return textInSent;
33 }

```

рис.4. Функція для перетворення тексту у масив строк

На рис.5-6 можна побачити функції класу Task

```

50- public static String[] findLongest(String[] inSentences, StringBuffer text) {
51    StringBuffer longest[] = new StringBuffer[inSentences.countSentences(text)];
52    for (int i = 0; i < inSentences.countSentences(text); i++)
53        longest[i] = new StringBuffer(0);
54    StringBuffer temp = new StringBuffer(0);
55    for (int i = 0; i < inSentences.countSentences(text); i++) {
56        int pos = -1;
57        int helper = 0;
58        String sentence = inSentences[i];
59        int j = 0;
60        int words = countSpaces(sentence);
61        while(j < words) {
62            for(int k = pos+1; k < sentence.length(); k++) {
63
64                char sign = sentence.charAt(k);
65                if (sign == SPACE) {
66                    pos = k;
67                    helper = k;
68                    break;
69                }
70                else
71                    temp=temp.append(sign);
72            }
73            if (longest[i].length() < temp.length()) {
74                longest[i]=longest[i].delete(0, longest[i].length());
75                longest[i].setLength(temp.length());
76                longest[i]=longest[i].insert(0, temp);
77                temp=new StringBuffer();
78            }
79            }else temp=new StringBuffer();
80            j++;
81        }
82    }
83    String[] result = new String[inSentences.countSentences(text)];
84    for (int i = 0; i < inSentences.countSentences(text); i++)
85        result[i] = longest[i].toString();
86    return result;
87 }
88

```

рис.5. Функція пошуку найдовшого слова

```

public static String[] findShortest(String[] inSentences, StringBuffer text) {
    StringBuffer shortest[] = new StringBuffer[InSentences.countSentences(text)];
    for (int i = 0; i < InSentences.countSentences(text); i++)
        shortest[i] = new StringBuffer("words");
    StringBuffer temp = new StringBuffer(0);
    for (int i = 0; i < InSentences.countSentences(text); i++) {
        int pos = -1;
        int helper = 0;
        String sentence = inSentences[i];
        int j = 0;
        int words = countSpaces(sentence);
        while(j < words) {
            for(int k = pos+1; k < sentence.length(); k++) {
                char sign = sentence.charAt(k);
                if (sign == SPACE) {
                    pos = k;
                    helper = k;
                    break;
                }
                else
                    temp=temp.append(sign);
            }
            if (shortest[i].length() > temp.length()) {
                shortest[i]=shortest[i].delete(0, shortest[i].length());
                shortest[i].setLength(temp.length());
                shortest[i]=shortest[i].insert(0, temp);
                temp=new StringBuffer();

                char sign = sentence.charAt(k);
                if (sign == SPACE) {
                    pos = k;
                    helper = k;
                    break;
                }
                else
                    temp=temp.append(sign);
            }
            if (shortest[i].length() > temp.length()) {
                shortest[i]=shortest[i].delete(0, shortest[i].length());
                shortest[i].setLength(temp.length());
                shortest[i]=shortest[i].insert(0, temp);
                temp=new StringBuffer();
            }
            else temp=new StringBuffer();
            j++;
        }
    }
    String[] result = new String[InSentences.countSentences(text)];
    for (int i = 0; i < InSentences.countSentences(text); i++)
        result[i] = shortest[i].toString();
    return result;
}

```

рис.6. Функція пошуку найкоротшого слова

### 3. РЕЗУЛЬТАТИ

На рис.7. можна побачити результати роботи програми

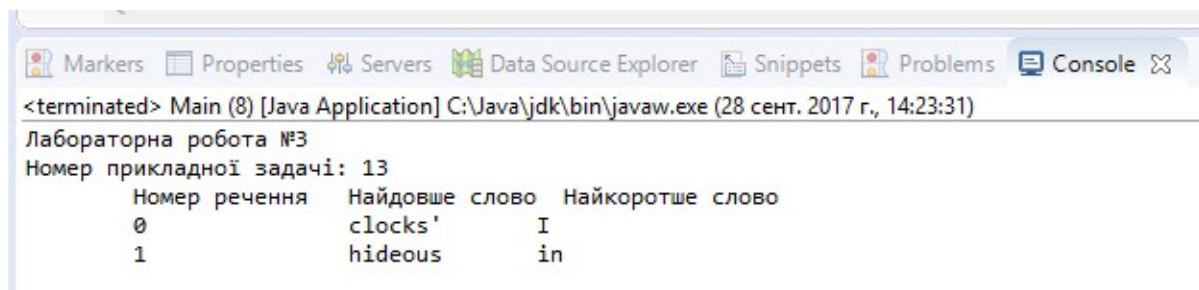


рис.7. Результати роботи програми

#### 4. ВИСНОВКИ

В результаті виконання лабораторної роботи ми навчилися розробляти утилітарні класи, здобули навички виконання прикладних задач за допомогою рядків та масивів.