Домашнє завдання №21

Застосовуючи JCL скласти програму (Java), яка за допомогою ***Collections.sort*** дозволяє з вхідного тексту(***String***) вивести слова, що починаються з заданої літери(решта слів вивести в алфавітному порядку). Сортування потрібно виконати без використання додаткової пам'яті, дозволяється використовувати тільки ***ArrayList***, що зберігає індекси на початок слів. Додатково потрібно зберегти результати роботи програми у ***HashMap***.

Застосувати різні способи для виводу результатів роботи програми.

** **коментар:** це завдання тотожне до завдань №19, №20 та №22(№22.1 і №22.2); таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції і запуску можна використати https://repl.it/languages/java або https://www.tutorialspoint.com/compile_java8_online.php .*

Вибір варіанту

Задана літера це перша літера прізвища студента *(записаного латинськими літерами)*

Приклад коду

Наведений зразок коду реалізовує завдання з виконання умови розміщення першими слів, що починаються на літеру 'К'.

Літера для прикладу	K
Декларація константи в класі	`final static char FIRST_CH = 'K';`

Лістинг

```
//// use default package // or use package: // package org.eom.acm;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Collectors;
import java.util.stream.Stream;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class Main {
  final static char FIRST_CH = 'K';
```

```
  final static int MAX_BUFFER_SIZE = 8192;

  public void scan(String str, ArrayList<Integer> list){
    if (str == null || list == null) {
      return;
    }

    Pattern token_re = Pattern.compile("[a-z]+", Pattern.CASE_INSENSITIVE);
    for(Matcher matcher = token_re.matcher(str); matcher.find();) {
      list.add(matcher.start());
    }
  }

  public static void copyStream(InputStream input, OutputStream output, int
start, int maxExpectedEnd) throws IOException{
        for(int index = 0; index < start; ++index) {
          input.read();
        }
        int maxExpectedBytesRead = maxExpectedEnd - start;
        byte[] buffer = new byte[maxExpectedBytesRead];
        for (int bytesRead; (bytesRead = input.read(buffer)) != -1 && bytesRead
<= maxExpectedBytesRead;){
           output.write(buffer, 0, bytesRead);
        }
  }

    public static String toString(Integer value) {
        return value.toString() + "\n";
    }

    public void printListIndexes(ArrayList<Integer> list) {
      if (list == null) {
        return;
      }
      try {
          // method 1(by to string function)
          copyStream(
              new ByteArrayInputStream(
              String.join(
                  "",
                  list
                  .stream()
                  .map(Main::toString)
                  .collect(Collectors.toList())
                  .toArray(new String[0])
                  ).getBytes()),
              System.out,
              0,
              MAX_BUFFER_SIZE);
          // method 2
          //list.forEach(System.out::println);
```

```
          // method 3
//System.out.print(list.stream().map(String::valueOf).collect(Collectors.joining
("\n")));
    } catch (IOException e) {
      e.printStackTrace();
    }
    }

    public void print(String str, ArrayList<Integer> list) {
      if (list == null) {
        return;
      }

    for (Integer value : list) {
      String word = str.substring(value);
      Matcher matcher = Pattern.compile("[a-z]+",
Pattern.CASE_INSENSITIVE).matcher(word);
      if(matcher.find()) {
        System.out.println(word.substring(matcher.start(), matcher.end()));
      }
    }
    }

    class ClassCompareFunction1 implements Comparator<Integer> {
      private String str;

      public ClassCompareFunction1(String text) {
          this.str = text;
        }

      Integer strcmp__withoutCase(int str1BaseIndex, int str2BaseIndex) {
        for (int str1Index = str1BaseIndex, str2Index = str2BaseIndex; str1Index
< str.length() && str2Index < str.length(); ++str1Index, ++str2Index) {
          int str1_tolower = Character.toLowerCase(str.charAt(str1Index));
          int str2_tolower = Character.toLowerCase(str.charAt(str2Index));
          if (str1_tolower != str2_tolower)
          {
            return str1_tolower < str2_tolower ? -1 : 1;
          }
        }
        return 0;
      }

      Integer strcmp_K__withoutCase(int str1BaseIndex, int str2BaseIndex) {
        int chr1_toupper = Character.toUpperCase(str.charAt(str1BaseIndex));
        int chr2_toupper = Character.toUpperCase(str.charAt(str2BaseIndex));
        if (chr1_toupper == FIRST_CH && chr2_toupper != FIRST_CH) {
          return -1;
        }
        else if (chr1_toupper != FIRST_CH && chr2_toupper == FIRST_CH) {
          return 1;
```

```
        }
        else if (chr1_toupper == FIRST_CH && chr2_toupper == FIRST_CH) {
          return strcmp__withoutCase(str1BaseIndex + 1, str2BaseIndex + 1);
        }
        return strcmp__withoutCase(str1BaseIndex, str2BaseIndex);
      }

      public int compareFunction(Integer arg1, Integer arg2) {
        //return str.substring(arg1).compareTo(str.substring(arg2)); // with
case sensitive
        return str.substring(arg1).compareToIgnoreCase(str.substring(arg2));
      }

      public int compareFunction1(Integer arg1, Integer arg2) {
        return strcmp_K__withoutCase(arg1, arg2);
      }

      public int compare(Integer arg1, Integer arg2){
            return compareFunction1(arg1, arg2);
        }
    }

    public void sort(String str, ArrayList<Integer> data){
      Collections.sort(data, new ClassCompareFunction1(str));
    }

    HashMap<Integer, String> getMapList(String str, ArrayList<Integer> list) {
      if (str == null || list == null) {
        return null;
      }

      HashMap<Integer, String> mapList = new HashMap<Integer, String>();

      for (int index = 0; index < list.size(); ++index) {
        String word = str.substring(list.get(index));
        Matcher matcher = Pattern.compile("[a-z]+",
Pattern.CASE_INSENSITIVE).matcher(word);
        if (matcher.find()) {
          mapList.put(index, word.substring(matcher.start(), matcher.end()));
        }
      }
      return mapList;
    }
    public static String toString(Map.Entry <Integer, String> data) {
        return "{ " + data.getKey().toString() + ", " + data.getValue() + "
}\n";
    }
    public void printMapList(HashMap<Integer, String> mapList) {
      if (mapList == null) {
        return;
      }
```

```
    // By "to string" function
      Stream.of(mapList.entrySet().toArray(new
Map.Entry[0])).map(Main::toString).forEach(System.out::print);
      //// By "to string" Lambda Expression
      //Stream.of(mapList.entrySet().toArray(new Map.Entry[0])).map(data -> "{ "
+ data.getKey().toString() + ", " + data.getValue() + "
}\n").forEach(System.out::print);
    }

  public static void main(String[] args) {
    Main acmhw21 = new Main();

    ArrayList<Integer> list = new ArrayList<Integer>();
    String text =
      "Sir, in my heart there was a kind of fighting " +
      "That would not let me sleep. Methought I lay " +
      "Worse than the mutines in the bilboes. Rashly- " +
      "And prais'd be rashness for it-let us know " +
      "Our indiscretion sometimes serves us well ... "
      ; // — Hamlet, Act 5, Scene 2, 4–8

    acmhw21.scan(text, list);
    acmhw21.sort(text, list);
    HashMap<Integer, String> mapList = acmhw21.getMapList(text, list);

    System.out.println("Indexes:");
    acmhw21.printListIndexes(list);

    System.out.println();
    System.out.println("Values:");
    acmhw21.print(text, list);

    System.out.println();
    System.out.println("Values(by map):");
    acmhw21.printMapList(mapList);

    // For suspend
    /*
    System.out.println("Press any key to continue . . . ");
    try {
      System.in.read();
    } catch (IOException e) {
      e.printStackTrace();
    }
    */
  }
}
```