PROGRAMMING TECHNIQUES

Documentation

2014-11-28

Erasmus Students
Karol Kalaga, Mateusz Łysień

Table of Contents

1. Task assignment

Our project was created by 2 Erasmus students – Karol Kalaga and Mateusz Łysień. The task assignment was follows:

- Karol Kalaga:
  - Implement all of algorithms in JAVA,
  - Design application structure,
  - Design UML diagrams.

- Mateusz Łysień:
  - Create project documentation
  - Design application structure,
  - Design application algorithms.

2. Domain (problem) analysis

Application can searching words in dictionaries. Application using 3 dictionaries, which contain 7600 files with words. Firs dictionary have files from 1 to 100. Second dictionary contain files from 1 to 1000. The Third dictionary contain all of the files – from 1 to 7600. The main problem is occurrence in chosen dictionary. The application must give absolute occurrence and relative occurrence. The absolute occurrence is a how many times putted word occurrence in all of the files – this is very simple to implement, because we must only count how many times this word occurrence. The relative occurrence is more difficult. We must get all of the occurrence word in chosen dictionary, this operation take more time, because must use few loops in the code to get count of occurrence word in dictionary. To searching we use full text searching. In the code we have 3 dictionaries represent as 3 SortedSet.

Full-text search works for us in a way that is charged current contents of the Text Field field is compared to all existing in the Structure of the dictionary and patterns that are most similar to the word given in the text field ARE placed in the table, according to the number of occurrences, however, the first criterion is the pattern of words, the second model the number of occurrences.

This is an example of source code :

```
private String FindText(String word, String pattern)
{
    int LenghtPattern = pattern.length();
    if (word.length() < LenghtPattern)
    return null;

   String tmp = word.substring(0, LenghtPattern);
   if (tmp.compareTo(pattern) == 0)
   return word;
    return null;
}
```

We have implemented a different type of search based on the method of searching for a needle in a haystack, in this version of the search, the needle is the word given in the text field while the structure is a stack of hay dictionary.

This is an example code :

```
 public String StrStr(String haystack, String needle)
{
    for (int i = 0; i < haystackLen; i++) {
      if (haystackLen - i + 1 < needleLen)
        return null;
     int k = i;
     int j = 0;
      while (j < needleLen && k < haystackLen && needle.charAt(j) == haystack.charAt(k)) {
        j++; k++;
        if (j == needleLen)
          Return haystack;
```

The next problem was to find and implement suitable for the needs of the data structure we used for this purpose the two structures to one dictionary, TreeMap and SortedSet. TreeMap proved to be reliable for the key word search of the word and collect the amount of occurrences, and we were able to quickly check the above collection is located has the word. Thanks Sorted Set we managed to sort the words according to the number of their occurrences. Helped into this comparator whereby adding another word to the Sorted Set it had been in place in line with the number of occurrences.

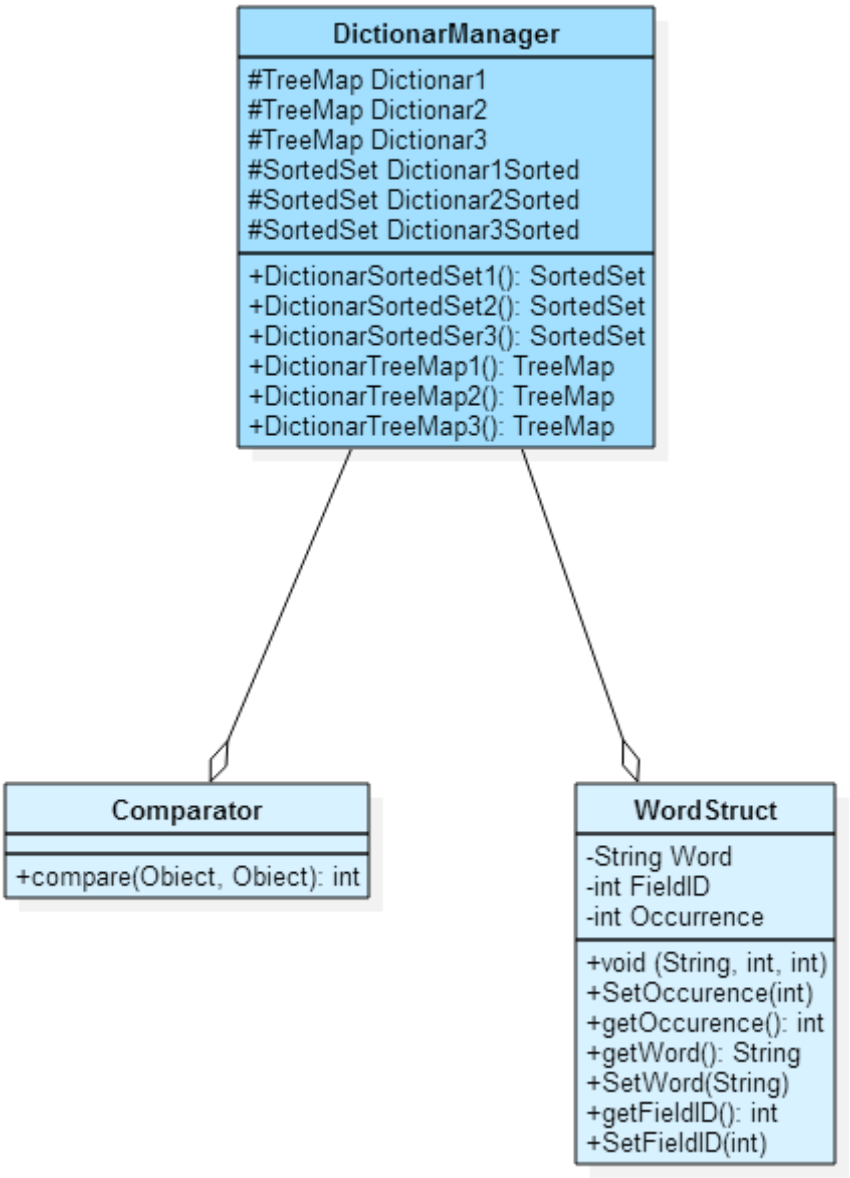
This is an example code of our Comparator :

```
@Override
    int compare(Map.Entry<String, WordStruct> o1, Map.Entry<String, WordStruct o2)
      if (o1.getValue().getOccurence() < o2.getValue().getOccurence())
        return 1;
      else
        return -1;
```

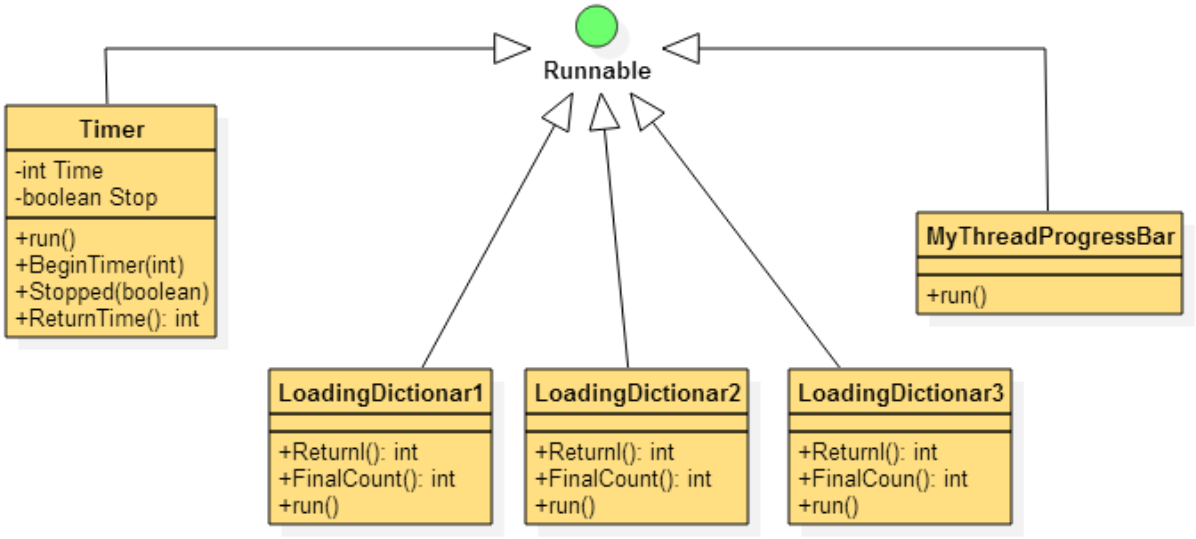
Another problem was that it took a long time to upload data to the structure of the file that is associated with many words there are more than 2.5 million, so we decided to complete the loading of data into separate threads, it provided us with two options: we could immediately after switching on with the program and the data that has already been loaded, and reduced time to load files from about 1 minute to 10-15 seconds.

3. Software design

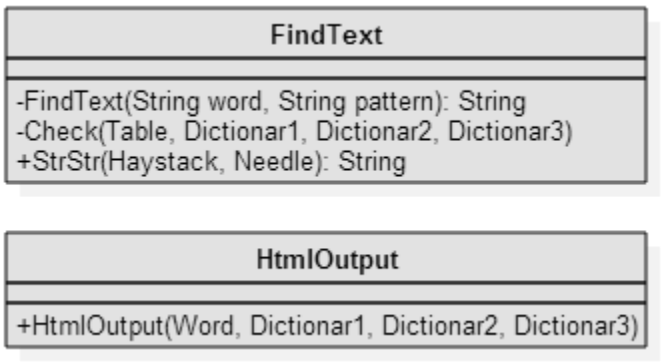
The main class DictionarManager stores data structure containing the words, divided according to the frequency of occurrences, using a comparator used connected to each structure SortedSet. The data are stored in a structure describing the features of a word that is of interest to us, the word, the word identification number and occurrence



Another group is a group class inherits from the interface Runnable, are grade three classes of parallel threads reading data structures, Timer which we used in testing to achieve the shortest time loaded words to structures and structures searching as soon as possible, and Thread supports the progress bar.

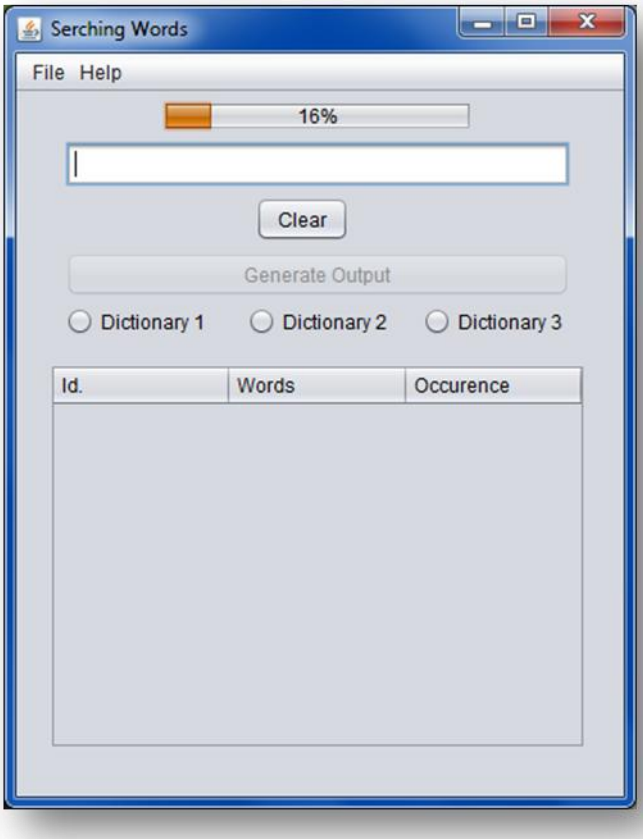


The last group are the classes that are used to search the structures and generating search results in the form of formatted output which used the EditorPane and the ability to add html.



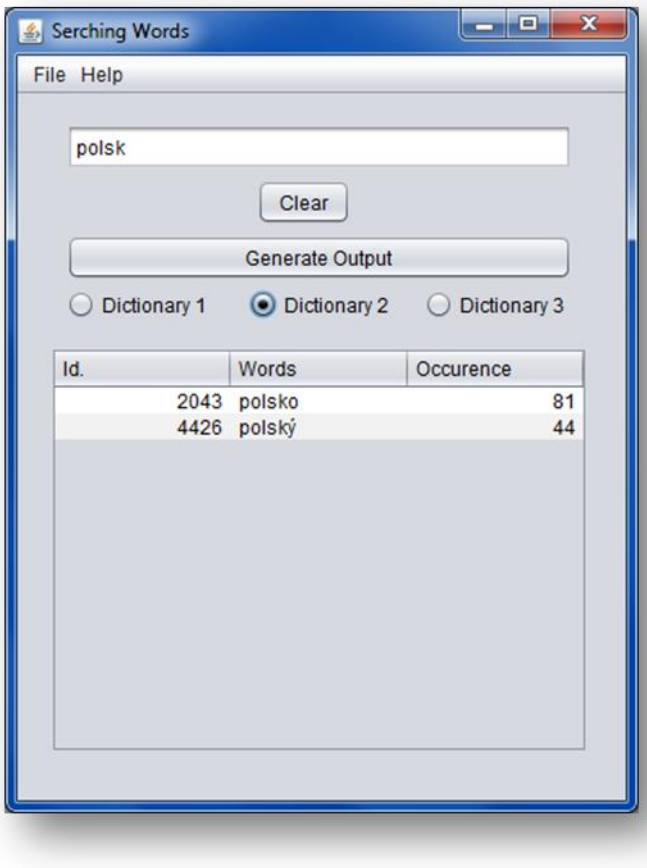
4. User manual

Our application is very simple to use, it's very intuitive.



Screen 1 Main application screen.

In the main application screen are a few option to use. In the top of the screen is a progress bar, which informing us how many percent files were loaded to the application. Loading all of the files take about 40-50 second. This progress bar will be remove when all of the files were loaded. When all of the files were loaded we can start using application. In the text box we can put some word, next we must declare in which dictionary we want search this word. Application use 7600 files. First dictionary use 1-100 files, Second dictionary use 1-1000, and Third dictionary use 1-7600 files.



Screen 2 Search result

When we declare in which dictionary we want to search, in the table will show search result. Result present word identifier, searching word, and word occurrence in chosen dictionary. The last option in the application is “Generate Output”. We use this option for generate HTML output with all information about searched word – absolute and relative occurrence in all dictionaries.

Screen 3 HTML output

5. Conclusion and work evaluation

Working with this project was opportunity to gain new experience. We learned how to work and how to using full text searching and how to utilize advanced data structures.

Involuntarily I also learned to utilize threads to speed up the program and a parallel problem solving. In addition, we learned to write your own class comparators for sorting, as in the situation when the value of a case is not the key, in the structure.

Thanks this project we get new skills in designing application, like projecting UML diagrams, thanks that we could had better start to implementing application. We spend quite a lot time for designing algorithms. Thanks this things we were very good prepared to implement code of application. The project was new experience for us, we thinking the project was done very good, we are feeling satisfaction for this project.