MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

Dictionary of synonyms (Thesaurus)

- Project #5-

Student: Dariana Lupea

Teacher: Delia Balaj

Group: 30425

Deadline: May 13, 2016

Table of contents

1. Introduction

1.1 Objective

Design *a Dictionary of Synonyms* for Romanian or English language. It is required to use Java Collection Framework Map for the implementation.

Problem specification

Consider the implementation of one of the following:

a) A dictionary of Romanian language or a dictionary of English language or

b) A dictionary of synonyms (thesaurus) for Romanian or English language.

It is required to use *Java Collection Framework Map* for the implementation. Define and implement a domain specific interface (populate / add / remove / copy / save / search, etc.). Consider the implementation of specific utility programs for dictionary processing. For example: - Implement a method for checking dictionary consistency. A dictionary is consistent, if all words that are used for defining a certain word are also defined by the dictionary. - Implement dictionary searching using * (any string, including null) and ? (one character). For example, you can search for a?t*. Use the above examples to warm up your imagination.

2. General aspects

2.1 Problem analysis

The application should implement all the requirements given in the task description. The given task has been interpreted in the form of a **Dictionary of Synonyms**. This Dictionary of Synonyms should provide users with the possibility of searching for words, adding new words to the thesaurus, removing unused words and listing all its content. All the information about the Dictionary's state is saved in a file after the user closes the application.

A **thesaurus** (plural "thesauri") is a reference work that lists words grouped together according to similarity of meaning (containing synonyms and sometimes antonyms),

in contrast to a dictionary, which provides definitions for words, and generally lists them in alphabetical order.

2.2 Modeling

One major advantage of having a dictionary of synonyms on the computer is the possibility of searching and finding quickly the word which is searched for. Moreover, a user will have the possibility to add his/her most often used words to the dictionary so that he/she can find them when they are needed. A thesaurus collection of words could be published also on the web, so that remote users can access it. At this point, the problem of protecting data arises, which in this small application has not been taken into account.

3. Projection

3.1. UML Diagrams

3.1.1 Use – case diagram

A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. This application can be used from two different perspectives: as a *regular user* or as a *dictionary manager*, so the corresponding use-cases are illustrated below:

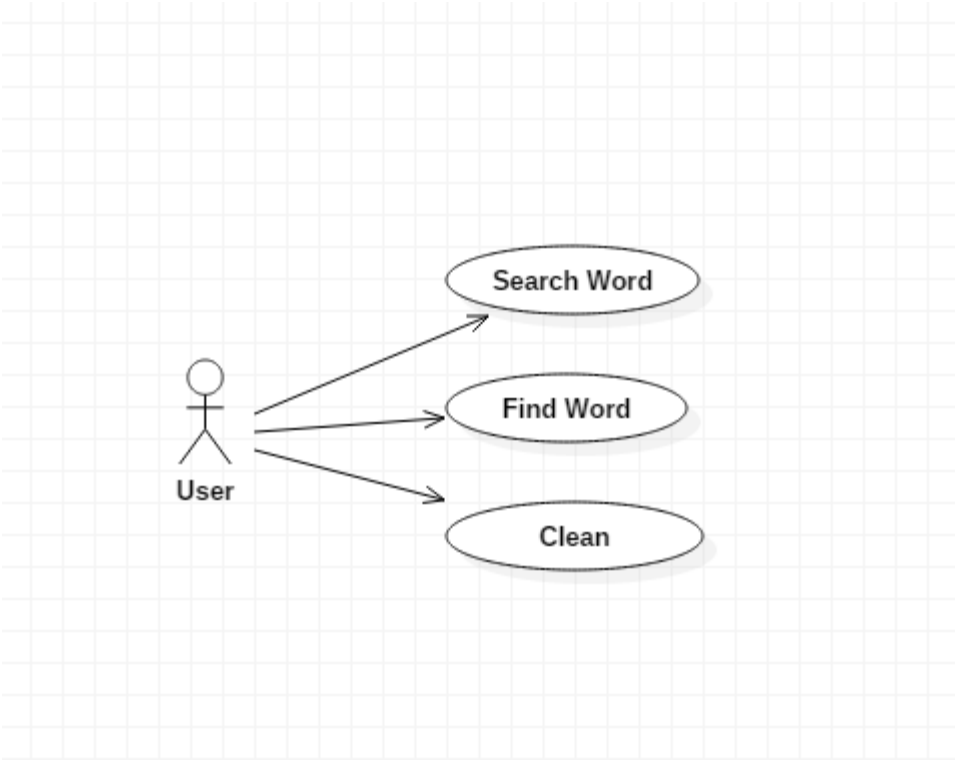


Fig. 1: Regular user: Use-case diagram

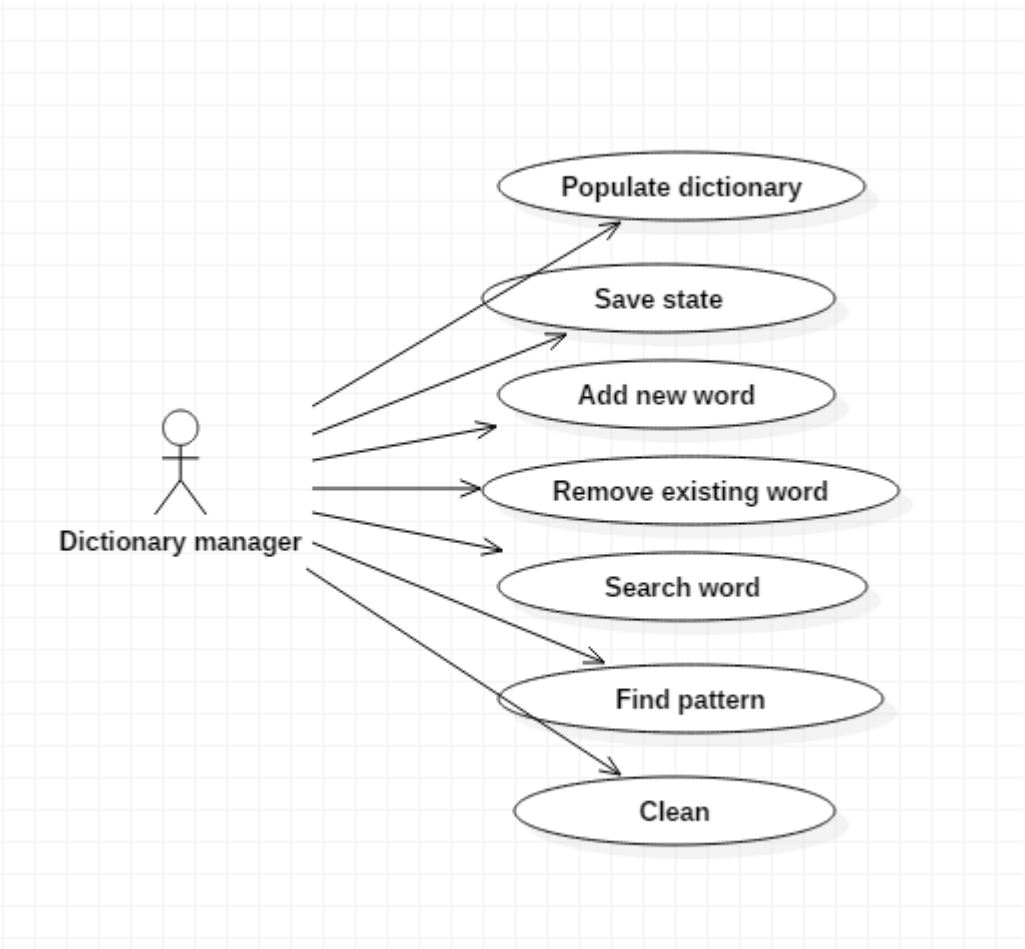


Fig. 2: Dictionary manager: Use-case diagram

3.1.2. Class diagram

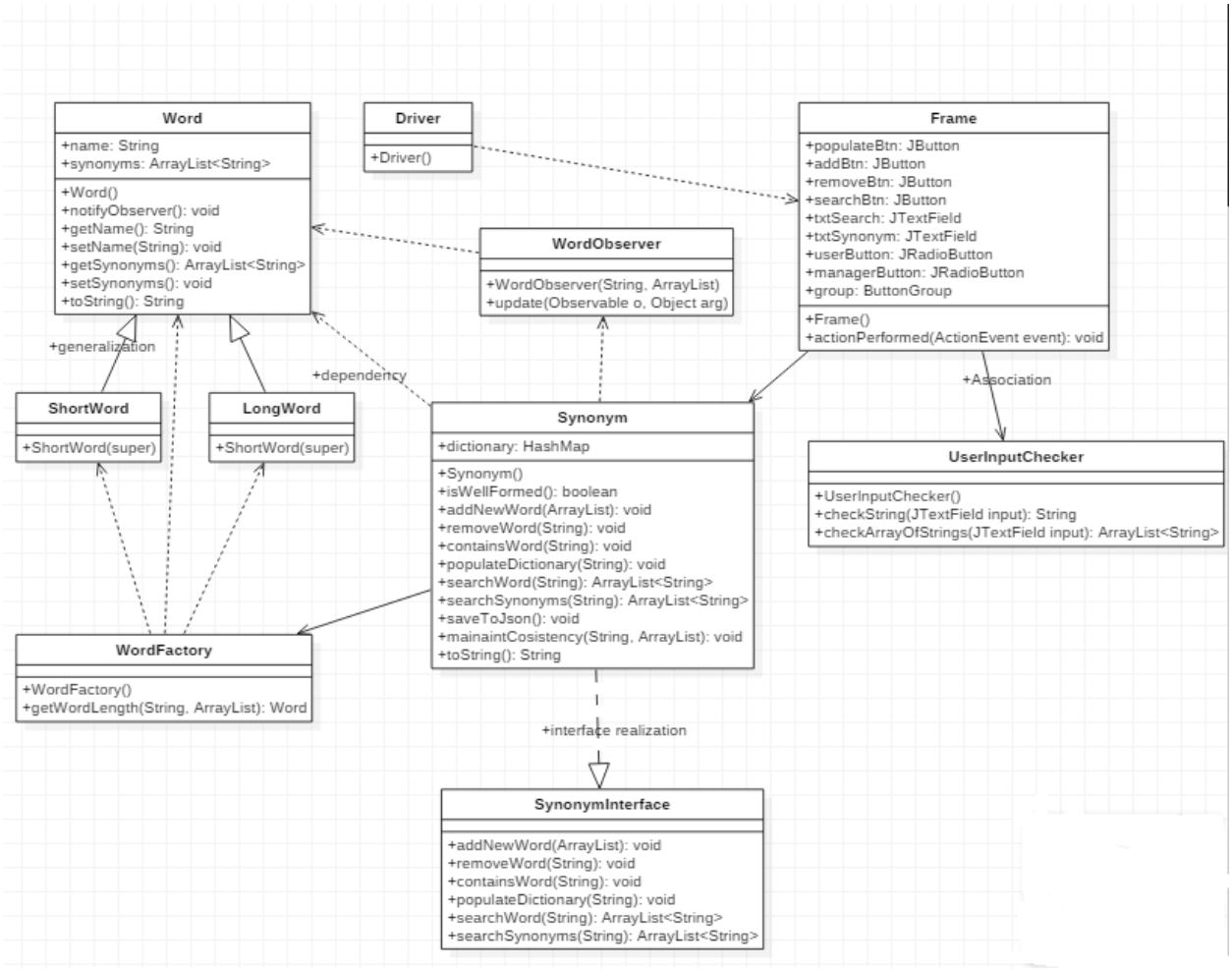


Fig. 3: Class diagram

3.1.3. Sequence diagram

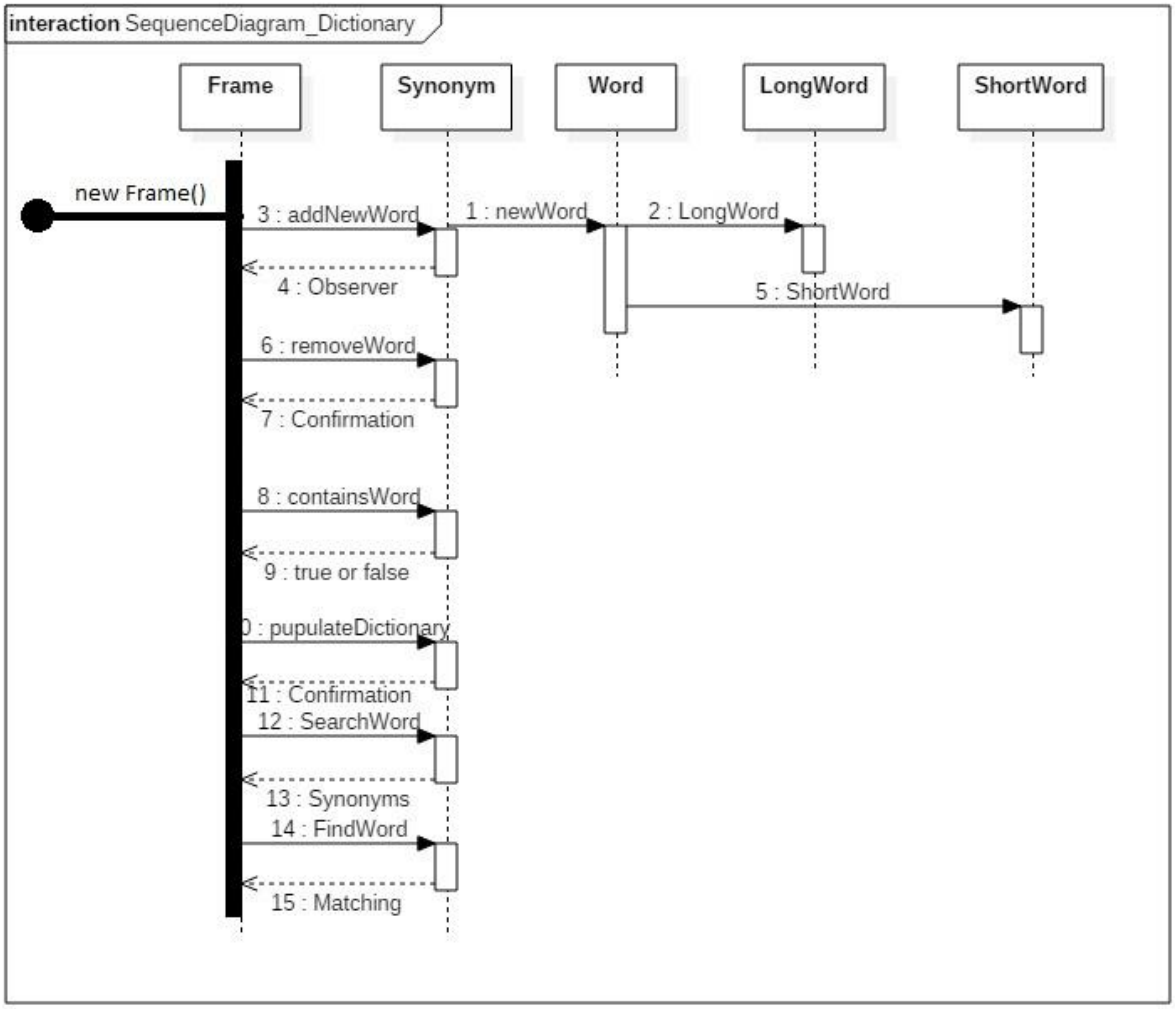


Fig. 3: Sequence diagram

3.1.4. Package diagram

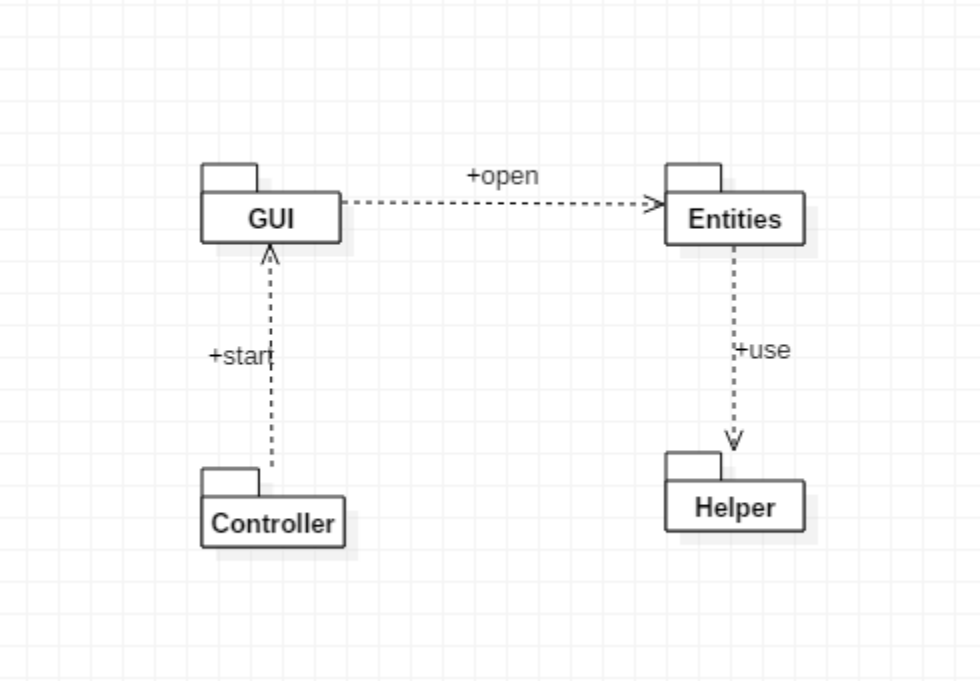


Fig. 5: Package diagram

3.1.5. Design Pattern diagrams

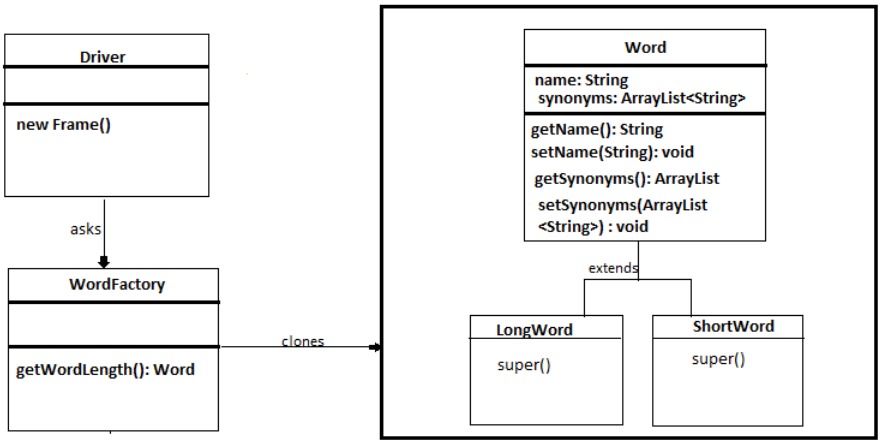


Fig. 6: Factory Pattern diagram

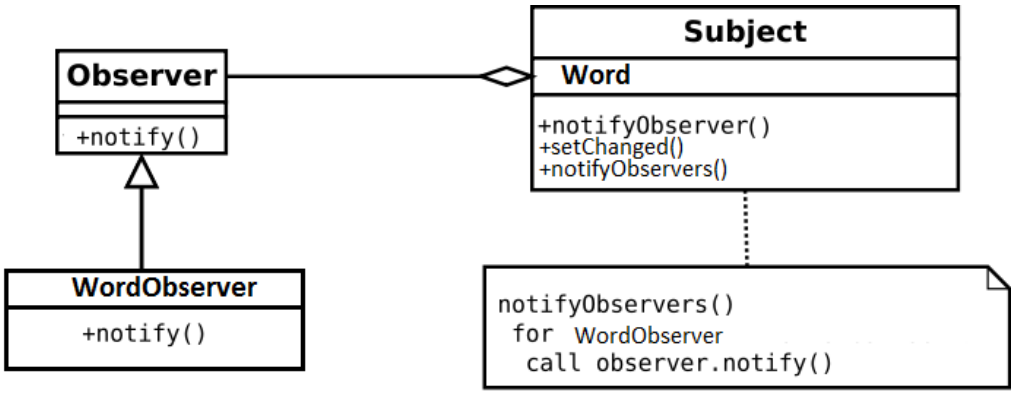


Fig. 7: Factory Pattern diagram

3.2 Data Structures

In this project implementation I have used several data structures, but the most significant one is:

- **HashMap**

Hash Map is actually Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

Advantage: This implementation provides constant-time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). It's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

Disadvantage: **Note that this implementation is not synchronized.** If multiple threads access a hash map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural

modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.)

Chosen data structure: Hash Map

Since this application will not use multithreading, there is no problem with using Hash Map. Moreover, we can come across the situation of permitting null entries. Another advantage of using Hash Map is that this collection is collision safe, because the entries of the table act like a linked list. When you put a new entry into the same bucket, it just adds to the linked list. If the hash of the key in the map collides with an existing key, the Map will re-arrange or keep the keys in a list under that hash. No keys will get overwritten by other keys that happen so be sorted in the same bucket.

3.3 Class Projection

After succeeding in analyzing the problem, we can go further to developing the classes that are necessary for the implementation of this application.

I obtained the final version of the system's structure. It contains 4 packages, namely: **Controller, Entities, GUI** and **Helper**. Each of these consists of other classes, which perform specific tasks. I will present them below:

- **Entities** package:
  - **LongWord** class
  - **ShortWord** class
  - **Word** class
  - **Synonym** class
  - **SynonymInterface** class
  - **WordFactory** class
  - **WordObserver** class

- **GUI** package:
  - **Frame** class – creates the user interface
- **Controller** package:
  - **Driver** class

- **Helper** package:

- **UserInputChecker** class

3.4 Packages

3.4.1 **Entities** package

It contains 7 classes, which capture the behavior of the application in terms of the problem domain: **LongWord**, **ShortWord, Word, Synonym, SynonymInterface, WordFactory** and **WordObserver**.

a) LongWord class

It models an entity of type **Word**. The length of a long word must be greater than 7. This class extends Word class.

b) ShortWord class

It models an entity of type **Word**. The length of a short word must be less than 7. This class extends Word class.

c) Word class

This class extends **Observable**. It contains a constructor and some getters and setters:

```
private String name;
private ArrayList<String> synonyms;

public Word(String name, ArrayList<String> synonyms) {
      this.setName(name);
      this.setSynonyms(synonyms);

}
```

d) Synonym class

This class could be considered the "main" one. It implements **SynonymInterface** interface, so it has to implement all the methods inherited, such as:

```
public void addNewWord(String word, ArrayList<String> synonyms);
public void removeWord(String word);
public boolean containsWord(String word);
public void populateDictionary(String fileName);
public ArrayList<String> searchWord(String pattern); etc.
```

e) **WordFactory** class

I have created this class in order to be able to use the Factory Design Pattern.

```
  public class WordFactory {

  public Word getWordLength(String word, ArrayList<String> synonyms) {
      if (word.length() < 7) {
            return new ShortWord(word, synonyms);
      } else {
            return new LongWord(word, synonyms);
      }
}
}
```

f) **WordObserver** class

It represents the **Observer** and it contains the **update** method.

```
 public class WordObserver implements Observer {


@Override
public void update(Observable word, Object arg1) {
      Word w = (Word) word;
      System.out.println("New word added:" + w.toString());

}

}
```

3.4.2 GUI package

It contains a class responsible with creating the user interface. I used **Swing**, which is a GUI widget toolkit for **Java:**

- **a top level container:** the class *Frame* extends **JFrame**

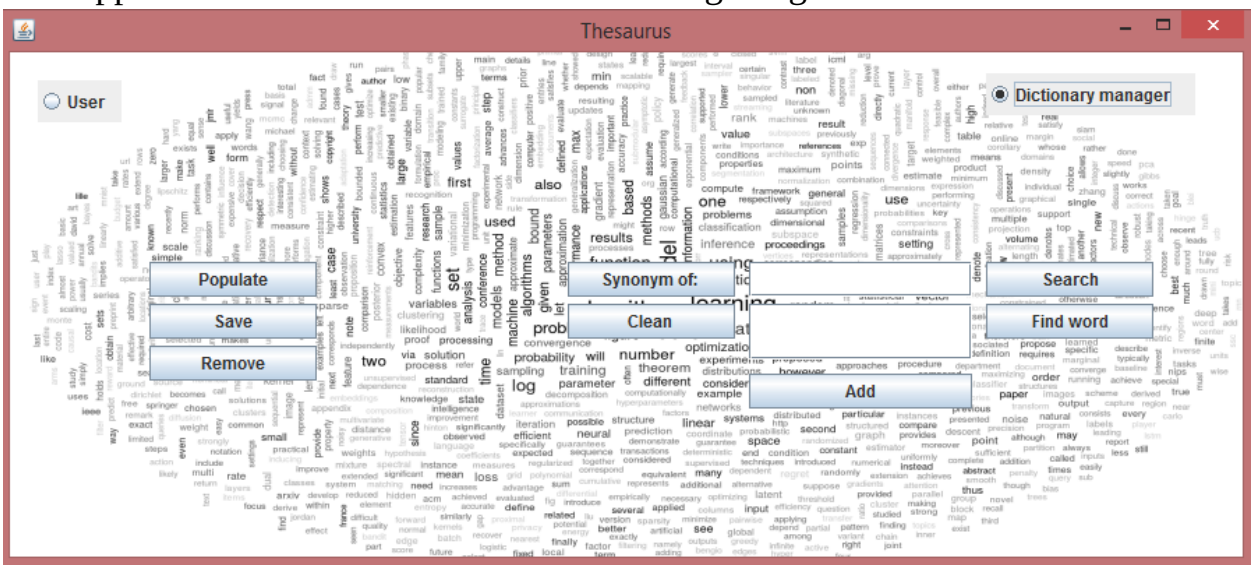
- **JComponents: JButtons, JLabels, JTextFields, JRadioButtons:**

```
private JButton populateBtn, addBtn, removeBtn, existsBtn, saveBtn,
searchBtn, synonymBtn, clearBtn;
private JTextField txtSearch, txtSynonym;

private JRadioButton userButton, managerButton;
private ButtonGroup group;
private JLabel backgroundLbl;
```

3.6 Interface

This application GUI looks like in the following image:



- The **user** can use this application in 2 different modes: as a Regular User or as a Dictionary Manager.

a) User

This type of user can perform the following operations: **Search, Find Word** and **Clean** the window**.** The user must:

- Enter the word to be searched
- Press “Search” in order to check for the synonyms
- Press “Search” in order to check the existence of a certain word
- Press “Clean” if he/she wants to clean the text area

4. Implementation and testing

Regarding the implementation process, I used the **Eclipse IDE**. During the program development steps, many changes were made and as I started to have more and more methods I realized the importance of a good structure. I refer here to organization of the code in packages and classes. I consider that the code I wrote is understandable and reusable. The algorithms I used are relatively easy, based on the well-known mathematical algorithms.

I have also tested each method, in the following way:

-> I used the console at first, for displaying the result

-> I initialized data with some appropriate values

-> I checked if the obtained result was the expected one

-> if YES, I continued the implementation process

-> if NO, I tried to figure out where is the error / problem coming from and solve it

5. Results

Concerning this project, I believe that it covers all the requirements from the problem specification and it fulfills the main purpose, which is performing some specific operations, in order to manage a bank.

6. Conclusion

To conclude, I can say that this project meant hard work, a lot of new things learned, focusing, development and creativity. Even if I encountered a lot of problems, I was able to fix them after all, by searching on the internet or asking a colleague for advice. I think that my application satisfies the requirements and the users will have at their disposal all its functionalities.

7. References:

- http://stackoverflow.com
- http://www.tutorialspoint.com/junit/junit_writing_tests.htm
- http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html
- http://www.tutorialspoint.com/java/java_serialization.htm