Programming Techniques

Laboratory Work - Homework 5

Name: Bologa Marius Vasile

Group: 30425

1.Project objectives

Objective

1. Study the Java Collection Framework Map
https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html

2. 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.

Note. The good things acquired as a result Homework 4 (i.e. contracts, invariants, assert, separating the interface from implementation, javadoc, etc.) will be also used for this homework.

2. Problem analysis , modeling scenarios , use cases

2.1 Problem analysis

The analysis of a problem starts from examining the real model or the model we confront with in the real world and passing the problem through a laborious process of abstractization. Hence we identify our problem domain and we try to decompose it in modules easy to implement. Starting from the concept of synonymus dictionary, I have started to underline the main, objects presented in the process of administrating a dicionary, the synonim, the word, the ArrayList of Synonyms. In order to be able to design and build a software program that performs and satisfies all of the specifications and requirements presented above it is very important to understand all the opperations needed in order to hava a sustainable application.

2.2 Modeling

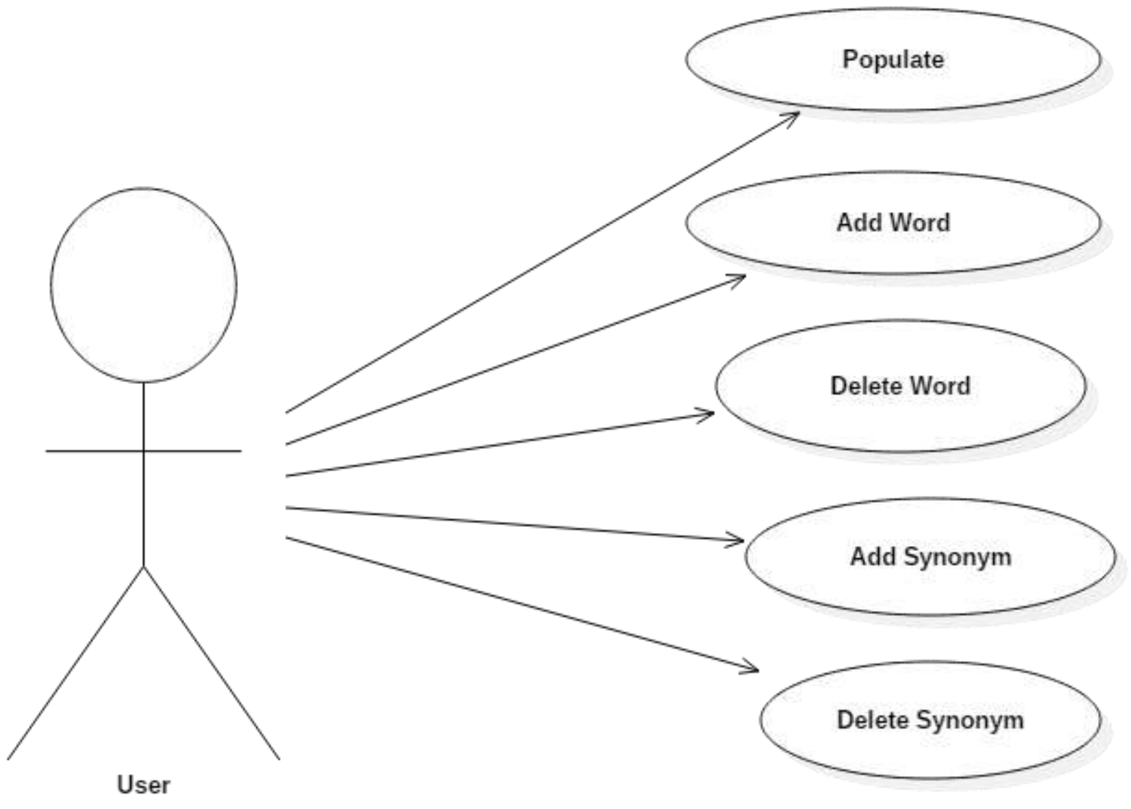
First of all, we need to find some means to store the Dictionaary.In order to save the accounts and information regarded to them, I have used the JSON files, in order to have my object written as a json file when the program ends, and to restore the dictionary at it's previous status, when the app was closed. I have used a HashMap<String,ArrayList<String>> in order to get the word and the synonyms of each word from the dictionary.

3. Projection and Implementation

In what the implementation is concerned this project was developed in Eclipse and it was only tested in this environment. However the program should maintain its portability. Concerning the code implementation I did not make use of laborious algorithms, but I have rather stayed faithful to the classical implementation of a dictionary.

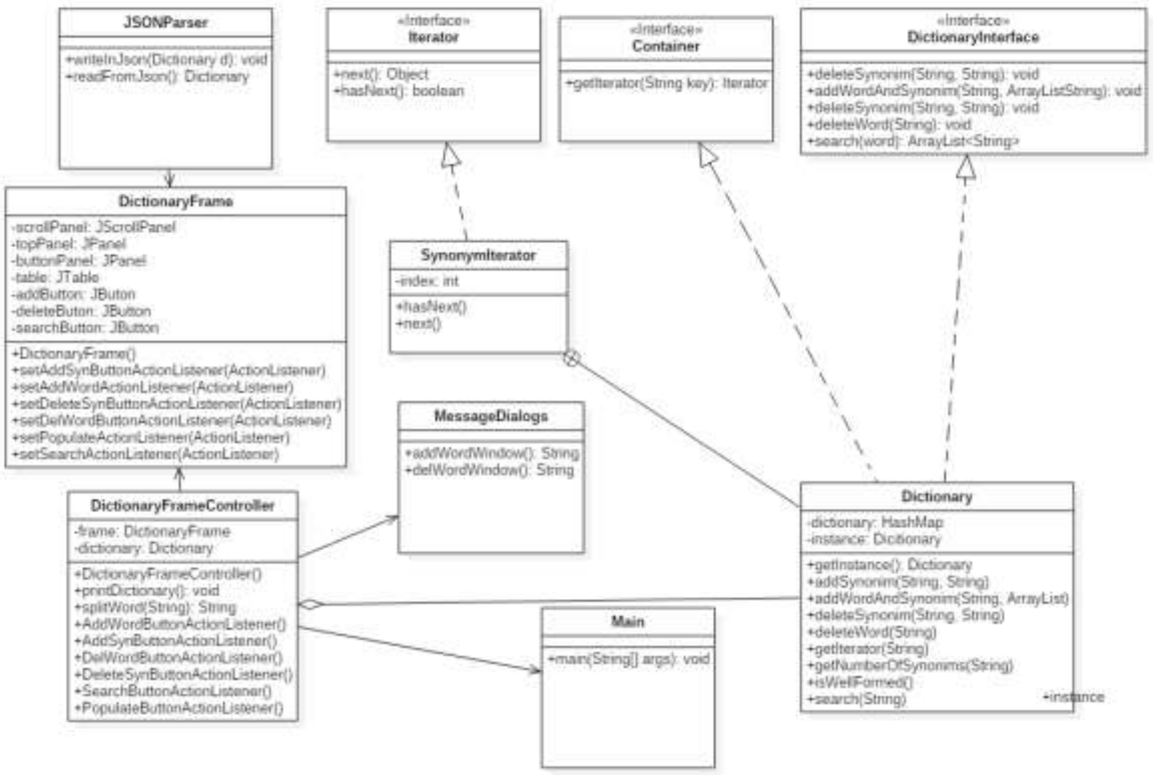
3.1 UML diagrams

3.1.1 Use case diagram

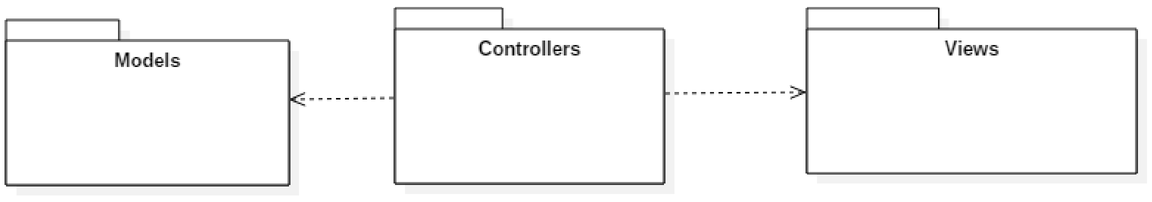


The use case diagram is nothing but the system functionalities written in an organized manner, presentig the actor, which is the admin in this case and the operations provided by the system for the user.

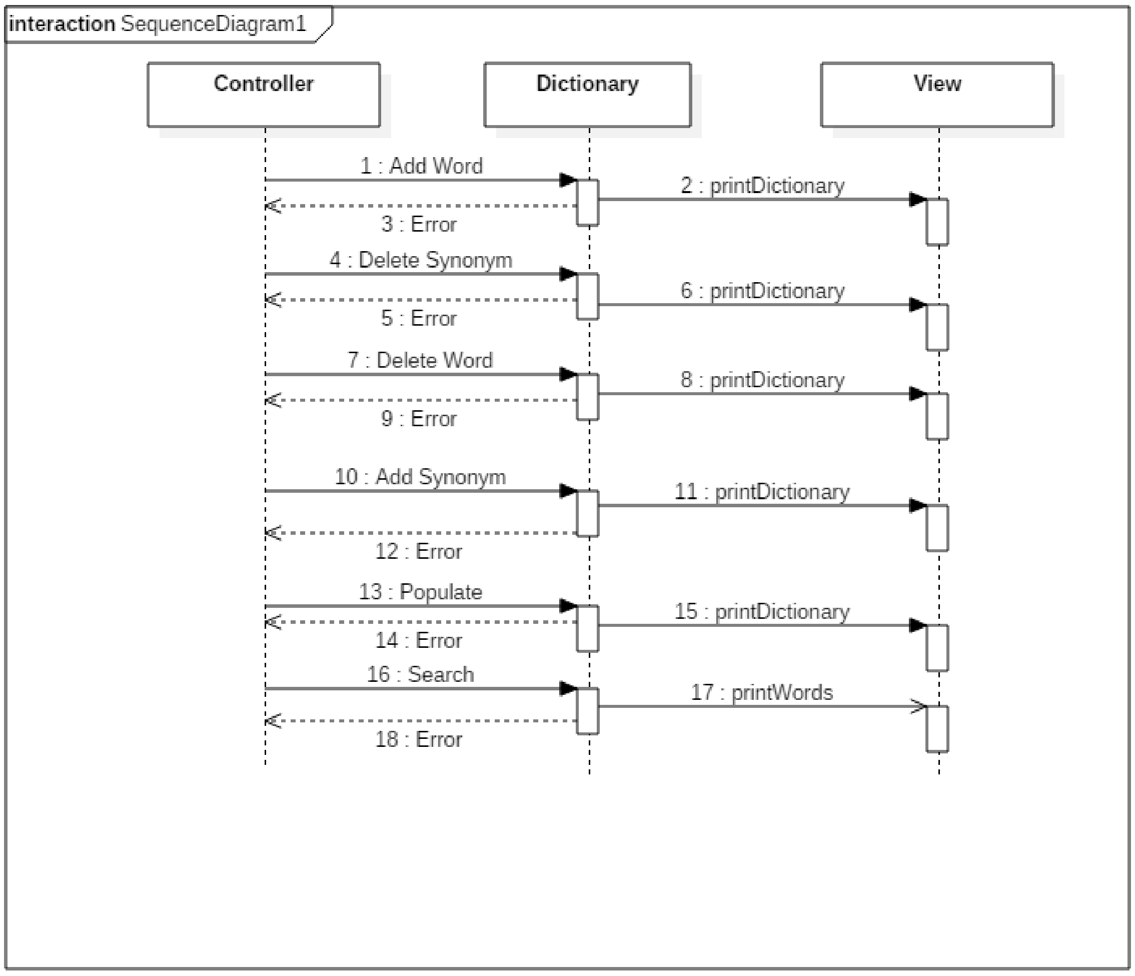
3.1.2 Class diagram


JSONParser
+writeInJson(Dictionary d): void
+readFromJson(): Dictionary
«Interface»
Iterator
+next(): Object
+hasNext(): boolean
«Interface»
Container
+getIterator(String key): Iterator
«Interface»
DictionaryInterface
+deleteSynonim(String, String): void
+addWordAndSynonim(String, ArrayListString): void
+deleteSynonim(String, String): void
+deleteWord(String): void
+search(word): ArrayList<String>
DictionaryFrame
-scrollPanel: JScrollPanel
-topPanel: JPanel
-buttonPanel: JPanel
-table: JTable
-addButton: JButon
-deleteButon: JButton
-searchButton: JButton
+DictionaryFrame()
+setAddSynButtonActionListener(ActionListener)
+setAddWordActionListener(ActionListener)
+setDeleteSynButtonActionListener(ActionListener)
+setDelWordButtonActionListener(ActionListener)
+setPopulateActionListener(ActionListener)
+setSearchActionListener(ActionListener)
SynonymIterator
-index: int
+hasNext()
+next()
MessageDialogs
+addWordWindow(): String
+delWordWindow(): String
DictionaryFrameController
-frame: DictionaryFrame
-dictionary: Dictionary
+DictionaryFrameController()
+printDictionary(): void
+splitWord(String): String
+AddWordButtonActionListener()
+AddSynButtonActionListener()
+DelWordButtonActionListener()
+DeleteSynButtonActionListener()
+SearchButtonActionListener()
+PopulateButtonActionListener()
Main
+main(String[] args): void
Dictionary
-dictionary: HashMap
-instance: Dictionary
+getInstance(): Dictionary
+addSynonim(String, String)
+addWordAndSynonim(String, ArrayList)
+deleteSynonim(String, String)
+deleteWord(String)
+getIterator(String)
+getNumberOfSynonims(String)
+isWellFormed()
+search(String)
+instance

3.1.4 Package diagram



3.1.3 Sequence diagram



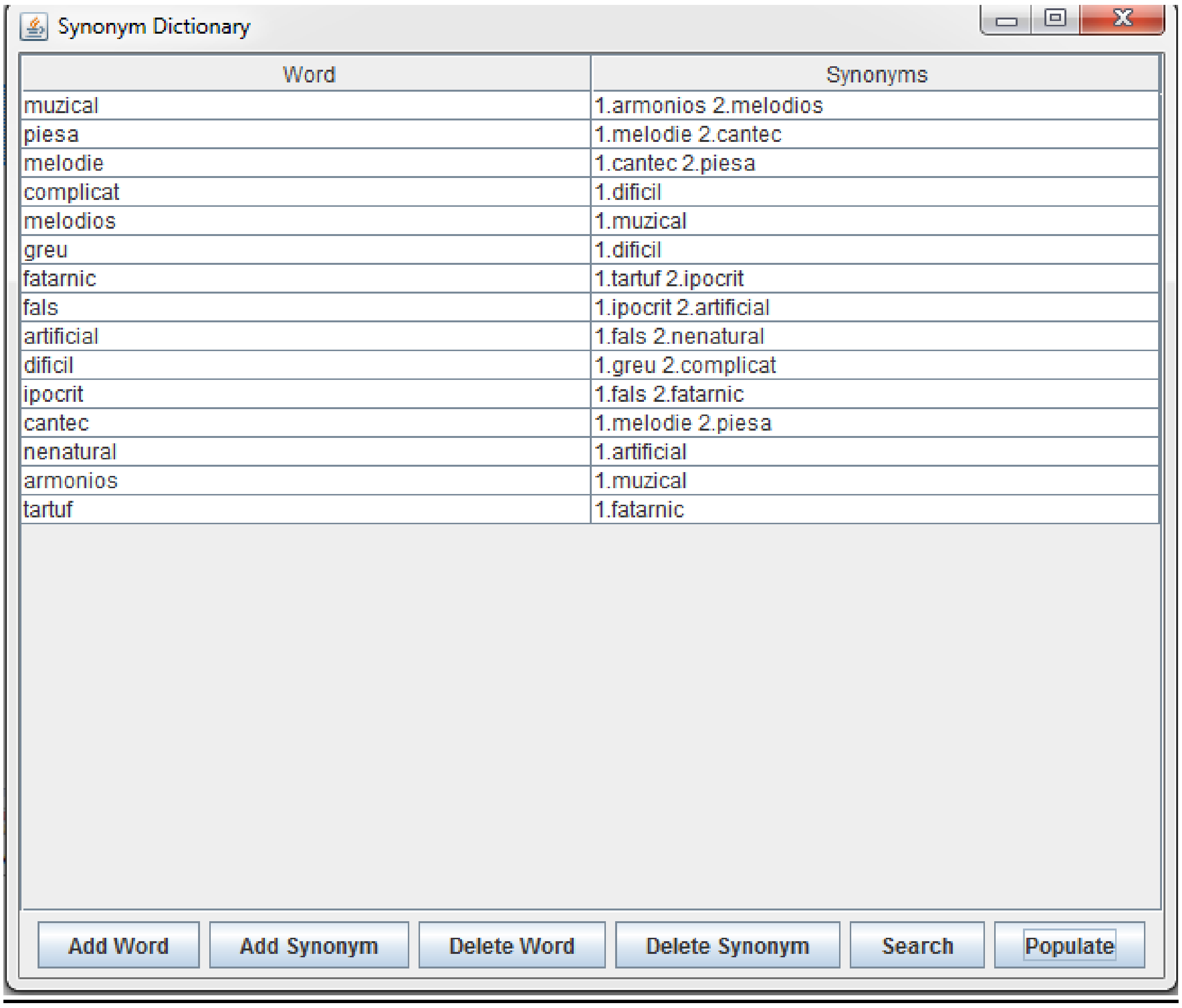
3.2 Data structures

The data structures used at this problem are either primitive data types such as integers or more complex data structures such as a HashMap or ArrayList<String>. The HashMap was introduced, to have a way to store the words and the synonyms of each word.

3.3 Class projection

Class projection refers mainly to how the model was thought, how the problem was divided in sub-problems, each sub-problem representing more or less the introduction of a new class. First I will start by mentioning exactly how my problem was divided into packages and afterwards each package with its own classes. I begin by creating the four packages I used: the first one being called „view” the second one being called „model”, and as a third one „controllers”. I named them intuitively because the first one handles the interface part, the part that deals with the user, the second one handles the implementation part, and the third one, is the one that contains the controllers of the interfaces. I will begin with the interface, then I will continue with the models and controller part.

3.4 Interface – package view



Word	Synonyms
muzical	1.armonios 2.melodios
piesa	1.melodie 2.cantec
melodie	1.cantec 2.piesa
complicat	1.dificil
melodios	1.muzical
greu	1.dificil
fatarnic	1.tartuf 2.ipocrit
fals	1.ipocrit 2.artificial
artificial	1.fals 2.nenatural
dificil	1.greu 2.complicat
ipocrit	1.fals 2.fatarnic
cantec	1.melodie 2.piesa
nenatural	1.artificial
armonios	1.muzical
tartuf	1.fatarnic

MessageDialogs.java – this class contains the main dialog windows that are created in order to get for exemple the name of the word searched, added or deleted. The class does not contain instance variables, and have only 2 methods:

public static String[] addWordWindow() : this method is used for creating a Joptionpane that will collect the data that is put into the fields of the pane, into an array of strings, from where we construct a word and add the new object to the Dictionary and to a table.

public static String[] deleteWordWindow():this method is used for creating a Joptionpane that will collect the data that is put into the fields of the pane, into an array of strings, from where we construct a word,and delete the new object from the dictionary and from the Table.

DictionaryFrame.java- this class represents the GUI of the application. The class contains the following instance variables:
private JScrollPane scrollPane: used especially for the tables to see the data easier
private static JTable *table:* a table where we have the contenst of the dictionary
private JButton addword, delWord, addSynonym, deleteSynonym, search,populate: buttons for the main operations that an admin can operate.

Constructor of this class places all the components enumerated above on the frame.
We have also getters and setters for the instance variables, and also we set the Listeners for the buttons.
public static JTable getTable()
public void setTable(JTable tableStock)
public final void setAddWordActionListener(**final** ActionListener a)
: add action listener for the add word button
public final void setDelWordActionListener(**final** ActionListener a)
: add action listener for the delete word button
public final void setAddSynonymButtonActionListener(**final** ActionListener a)
: add action listener for the add synonym button
public final void setDeleteSynonymButtonActionListener(**final** ActionListener a)
: add action listener for the delete synonym button
public final void setSearchActionListener(**final** ActionListener a)
: add action listener for the search button
public final void setPopulateActionListener(**final** ActionListener a)
: add action listener for the populate button
public static JTable getTableOrder()
public void setTableOrder(JTable tableOrder)

3.5 Models

3.5.1 Models

Dictionary.java- this class represents the content of the dictionary (the word / synonymd) . The dictionary is represented as a HashMap that has a word as keys and a list of words as values.The constructor :

The following methods are from the Dictionary class, and for the first method I left the implementation of the assertion for the pre/post conditions.

```
public void addSynonim(String word, String synonim) {
            assert isWellFormed();
            assert word != null;
            assert synonim != null;
            int n = getNumberOfSynonims(word);
            if (dictionary.containsKey(word)) {
                  dictionary.get(word).add(synonim);
            }
            if (dictionary.containsKey(synonim)) {
                  dictionary.get(synonim).add(word);
            } else {
```

```
                        ArrayList<String> s = new ArrayList<String>();
                        s.add(word);
                        dictionary.put(synonim, s);
                }
                assert (getNumberOfSynonims(word) != 0) && (getNumberOfSynonims(word)
== n + 1);
                assert dictionary.get(word).contains(synonim);
                assert isWellFormed();
        }

        public void deleteSynonim(String word, String synonim)
        public void addWordAndSynonim(String word, ArrayList<String> ss)
        public void deleteWord(String word)
        public ArrayList<String> search(String s)
        public int getNumberOfSynonims(String word)
        public boolean isWellFormed()
        public models.Iterator getIterator(String key)

        private class SynonymsIterator implements models.Iterator
                        @Override
                public boolean hasNext()

        public Object next()
```

<u>DictionaryInterface.java- </u> the interface DictionaryInterface contains the main functions that a bank can have like addWord, deleteWord, addSynonym(), deleteSynonym(), search() and for each of this function are defined preconditions and postcondition definede as comments and implemented using assert instruction in Dictionary class. Also each method have defined an Invariant for the dictionary class method isWellFormed (will be used as both precondition and postcondition for each method in Dictionary). Methods from the interface are listed below and for the first function there are presented the pre/post conditions and the invariant.

```
/**
         * Method for adding a new synonym
         *
         * @invariant isWellFormed()
         * @pre word != null && synonym!=null
         * @post getNumberOfSynonims(word)!=0 && getNumberOfSynonims(word) ==
getNumberOfSynonims(word)@pre + 1
         * @post get(word).contains(synonim)
         * @invariant isWellFormed()
         * @param word
         * @param synonym
         *
         */

        public void addSynonim(String word, String synonym);
        public void deleteSynonim(String word, String synonim);
        public void addWordAndSynonim(String word, ArrayList<String> ss);
        public void deleteWord(String word);

        public ArrayList<String> search(String s);
```

3.5.2Iterator

```
public interface Iterator {
	public boolean hasNext();

	public Object next();

}
```

3.5.3Container

```
public interface Container {
	public models.Iterator getIterator(String key);
}
```

3.6 Controllers

JSONParser.java- the role of this class is to save the class dictionary in a JSON file type when the program ends, and to restore them when the application starts again. This class has no instance variables, but has 2 methods:

public static Dictionary readFromJson():this method will take from a JSON file the contents and creates an object of type dictionary that is returned by the method

public static void writeInJson(Dictionary d): this method will take as an argument the class Dictionary that needs to be saved in a JSON file, and write it's content to an output json file

DictionaryFrameController.java- this class creates the frame DictionaryFrame that we created in the package View and adds ActionListener to the buttons to handle the events given. To do that, we create classes AddwordListener, DeletewordListener, AddsynonymListener, DeletesynonymListener, GenerateReportsListener that implements ActionListener. This implies the implementation of the methos actionPerformed.

```
public DictionaryFrameController() {
frame.setAdWordActionListener(new AddButtonActionListener());
frame.setDelWordButtonActionListener(new DeleteButtonActionListener());
frame.setAddSynonymButtonActionListener(new AddSynonymButtonActionListener());
frame.setDeleteSynonymctionListener(new DeleteSynonymButtonActionListener());
frame.setSearchButtonActionListener(new SearchButtonActionListener());
this.bank=manager.deserializeBank();
}
```

MainController.java

The most important thing from this class is the main function that calls the function of creating a new DictionaryFrameController that has as an instance variable a frame of type DictionaryFrame window, from where the application starts:

public static void main(String[] args) {

```
            new DictionaryFrameController();
    }
```

5. Results

The application is an user friendly and useful application to perform basic management operations such as: adding a word, delete a word, delete synonym, add a synonym, search, populate. Output results of the application are actually the results of operations performed on the list of word and list of synonyms.The information and the data are displayed in a table, as can be seen from the picture that shows the graphical user interface.The results of every action that it is done upon the word/synonyms can be seen in the tables. Even though being limited, this application can be considered a good and helpful tool in understanding the operations that stand behind a dictionary.

6. Conclusion and future developments

Achieving such a program may be hard both in terms of algorithms, graphical structure. The best is to represent the synonyms of a word as a tree type structure because this kind of structure makes it easier for some operations to be done: add, remove or search for an element from the structure.

For a better performance there should be implemented all cases where exceptions can occur and the application stops working due to an error made by the user or the admin. Also, this project could be implemented with a data base in the backround, to be more close to the reality and usefulness of this kind of applications.

7. Bibliography

http://stackoverflow.com/
http://www.json.org/ https://sites.google.com/site/gson/gson-user-guide
http://wiki.fasterxml.com/JacksonInFiveMinutes
http://www.tutorialspoint.com/design_pattern/