

PROGRAMMING TECHNIQUES

Homework 5

Student : Tudor Vlad Bresan

**Group:** 30423

Table of Contents

[Homework 5 1](#_Toc450876539)

[1 Problem specification 4](#_Toc450876540)

[2 Problem analysis, modeling, scenarios, use cases 4](#_Toc450876541)

[2.1 Problem analysis 4](#_Toc450876542)

[2.2 Modeling 5](#_Toc450876543)

[2.3 Scenario and use cases 5](#_Toc450876544)

[3 Design 6](#_Toc450876545)

[3.1 Class Diagram 6](#_Toc450876546)

[3.1.1 Dictionary Proc 7](#_Toc450876547)

[3.1.2 Dictionary 7](#_Toc450876548)

[3.1.3 GUI Class 7](#_Toc450876549)

[3.1.4 Controller 7](#_Toc450876550)

[3.1.5 View 7](#_Toc450876551)

[3.1.6 Serialization 7](#_Toc450876552)

[3.1.7 Serialization JSON 7](#_Toc450876553)

[3.1.8 Serialization Ser 7](#_Toc450876554)

[3.2 Package diagram 8](#_Toc450876555)

[3.3 Sequence diagram 8](#_Toc450876556)

[3.4 Data Structures 9](#_Toc450876557)

[3.5 Relationships 9](#_Toc450876558)

[3.6 User Interface 10](#_Toc450876559)

[4 Design Patterns 10](#_Toc450876560)

[4.1 Singleton pattern 10](#_Toc450876561)

[4.2 Model View Controller pattern 11](#_Toc450876562)

[4.3 Template pattern 12](#_Toc450876563)

[5 Implementation 13](#_Toc450876564)

[6 Results 17](#_Toc450876565)

[7 Conclusion and future development 17](#_Toc450876566)

# Problem specification

Consider the implementation of one of the following:

1. A dictionary of Romanian language or a dictionary of English language
2. 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\*.

# Problem analysis, modeling, scenarios, use cases

## Problem analysis

As mentioned above , in the problem specification area, we had a few options of what we are able to implement. For this project , I have chose to implement a dictionary of synonyms. At first it seemed like a great idea but as I went deeper and deeper into it’s logic I’ve encountered some problems which were all finally sorted out.

Firstly I’ve decided to store the words and their synonyms in files. Thus I’ve chosen an OOP style which allowed me to user serializable files. The data stored in these files is structured as follows: we have a Map which has String keys and a set of String values. The keys represent the the words that are going to be defined and the set of Strings ( the values ) are going to be the key’s synonyms.

The project will allow the user to perform operations like : adding a new word, deleting an existing word, updating an existing word’s synonyms , searching for a word in the dictionary etc.

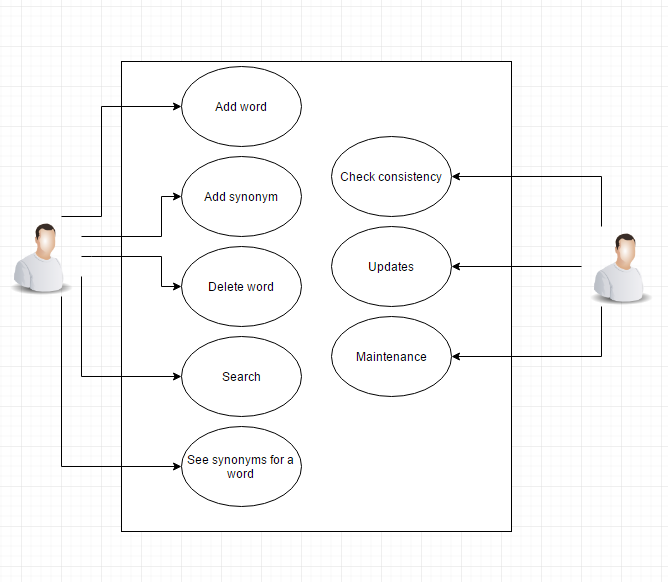
Furthermore , after doing some research on the internet and reading some chapters from computer science books that might have helped me , I have decided that the best way to provide the information requested by the user is by displaying the results in tables. These tables are going to be present the in the classes which sum up the graphic user interface

## Modeling

The implementation of this project is going to be done as follows :we’ll have a class that is responsible of reading and writing into a .ser file and as well as in a .json file, an interface called DictionaryProc which will contain the main methods to be applied in a dictionary such as: add a new word , delete an existing node , add a new synonym , search for a word etc. ; the Dictionary class will implement the methods declared in the interface ; the Serializable class as an abstract class which will be the super class for the Serializable JSON class and the Serializable Ser class; and the classes that will make the interaction with the application possible , the controller class which implements action listeners for each button, text field etc. the GUI class which will initialiaze the main graphic user interface ; the View class which is going to make the visualization of results possible and the main class which is going to start up the program.

## Scenario and use cases

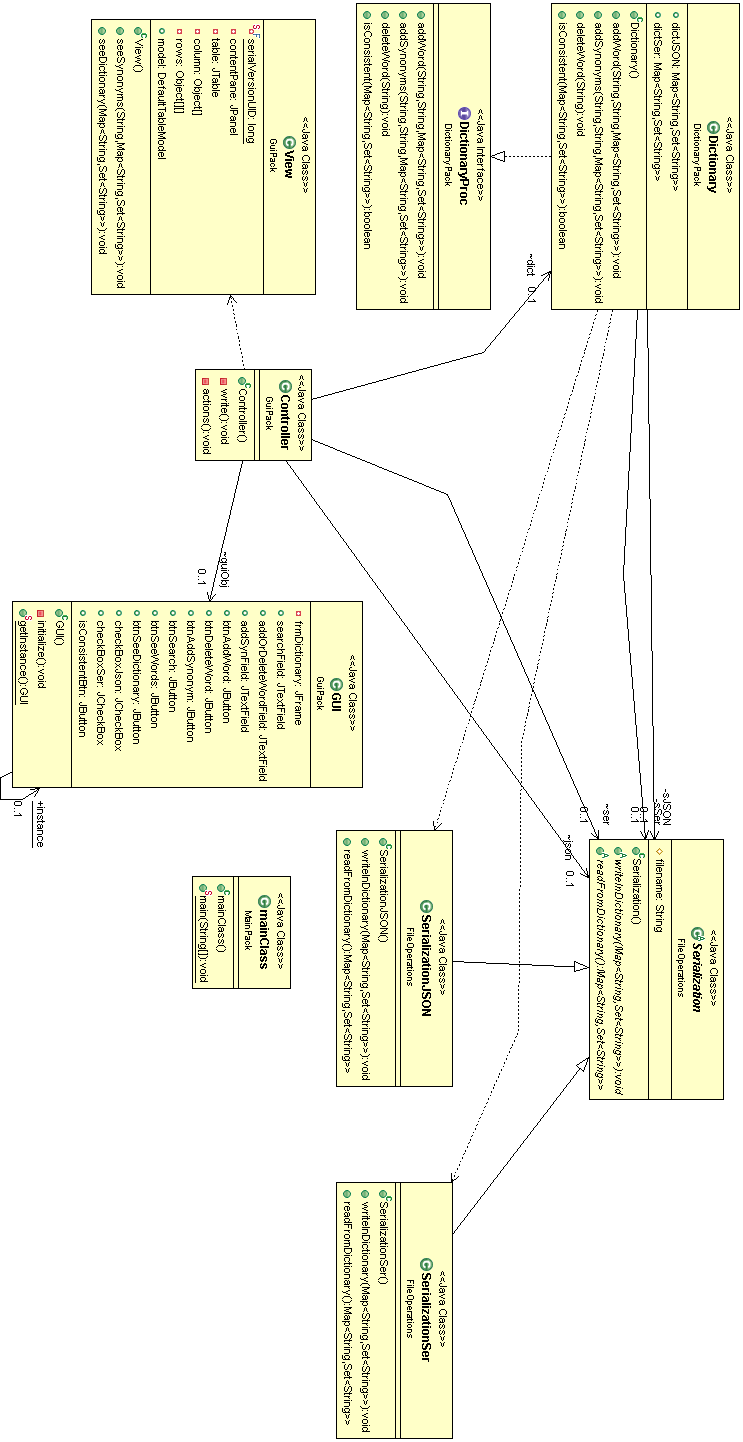
The user will be able to do some actions like : adding a new word , deleting an existing word , seeing the full content of a dictionary ( the user can chose which dictionary to see since we have 2 of them ) , add a new synonym to an existing word, search for a word in one of the dictionaries or in both of them , see the synonyms of an inputed word . In order to do any operation the user must enter the necessary data. The data that can be entered are : the word to be added , the synonym to be added or the word to be searched in the desired dictionary.



# Design

## Class Diagram

The UML diagram is a class diagram in which we can find the relationship between classes and also the elements(class variables , methods) a specific class contains.



### Dictionary Proc

The Dictionary Proc , as the name is suggesting , it’s showing the “behavior” of the class Dictionary .All the methods with empty bodies are going to be implemented in the class mentioned here (3.1.1)

### Dictionary

The Bank class implements all the methods declared in the Bank Proc interface. The methods implemented by this class are: add word , add synonym , delete word , is Consistent.

### GUI Class

This class is designed to create a graphical user friendly interface to simplify the use of the application. This class is a subclass JFRame for the purpose of using objects like : buttons , frames, textfields, panels.

In the graphic user interface we can observer text fields which are as follows the word, the synonym and the word to be searched in the dictionary. Furthermore it initializes the buttons for which we will create action listeners int the Controller class. The buttons are : add word , delete word , see dictionary , add synonym , is consistent, search , and two check boxes which will server as a decision of which dictionary is going to be used.

### Controller

In this class we will implement the action listeners for each text field , button , label and tables . It also contains methods which help us filter the results. It uses an object of class GUI and of class Bank

### View

The view class will be the main class where the results we be presented to the user by adding them into a JTable . This class opens a new frame each time the user searches for a word , wants to see the synonyms of a word .

### Serialization

The serialization class is an abstract class which is going to describe the main purpose of a Serialization file meaning the reading and the writing . It also contains a String in which we are going to store the name of the file . The file name will be the same but the extension of the two files which are going to be created will be different.

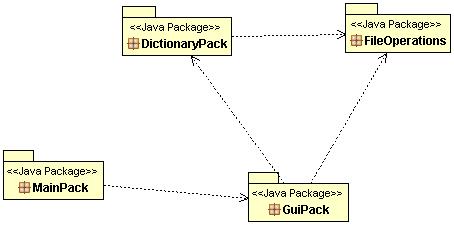
### Serialization JSON

This class is a subclass of the Serialization class. It implements the methods write In Dictionary and read From Dictionary which are going to be specific to the file with extension .json.

### Serialization Ser

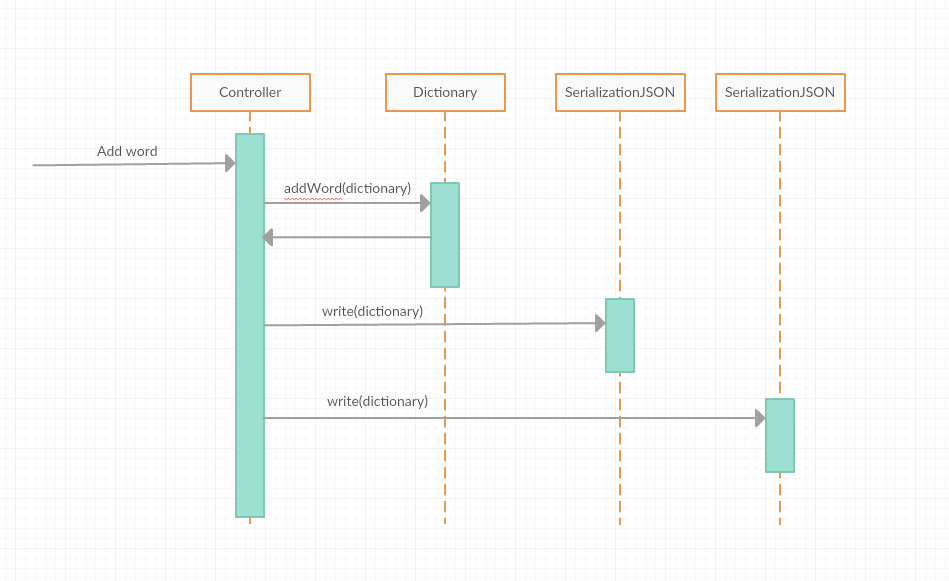
This class is a subclass of the Serialization class. It implements the methods write In Dictionary and read From Dictionary which are going to be specific to the file with extension .ser .

## Package diagram



These packages contain classes that implement specific methods. Each package has it’s role and will not implement any other methods that are not related to the others.

## Sequence diagram



The above picture represents an example of how the objects communicate between each other.

## Data Structures

The main data structure used in this project is the HashTable. In computing, a hash table (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Ideally, the hash function will assign each key to a unique bucket, but it is possible that two keys will generate an identical hash causing both keys to point to the same bucket. Instead, most hash table designs assume that hash collisions—different keys that are assigned by the hash function to the same bucket—will occur and must be accommodated in some way.

In a well-dimensioned hash table, the average cost (number of instructions) for each lookup is independent of the number of elements stored in the table. Many hash table designs also allow arbitrary insertions and deletions of key-value pairs, at (amortized) constant average cost per operation.

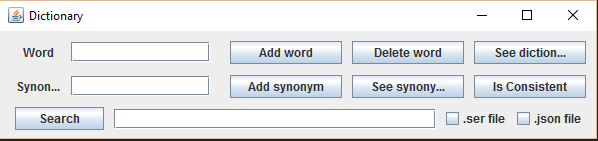
In many situations, hash tables turn out to be more efficient than search trees or any other table lookup structure. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and sets.

## Relationships

To solve the problem specifications the program uses 8 classes and 1 interface , so it can be more clear and be understood better . The name of the classes are representative and they do exactly what they’re supposed to be doing .

There are dependencies between the classes of this project, which means that a class uses another class methods or objects . The hierarchy is at follows : the class main creates a new Controller object which means it uses the Controller class. The controller class will initialize the graphic user interface . Each time a button is pressed on that interface an action listener will be ran from the Controller class. In that action listener we’ll have pieces of code that are specialized in executing the commands that are supposed to be executed when a specific button is pressed . That being said , a link to the other classes appear,Controller uses methods from the 3 classes that implement the operations , Serialization Ser , Serilation JSON and Dictionary . The reading and writing methods are created in the FileOpertaions package which is specialized in file operations.

## User Interface



I’ve tried making the user interface as friendly as possible but it could use some aesthetic improvements.

# Design Patterns

In order to create a program with a good OOP concept I’ve chosen some design patterns .The design patterns which are use in this project are : Singleton pattern , Model View Controller pettern and the template pattern.

## Singleton pattern

Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.



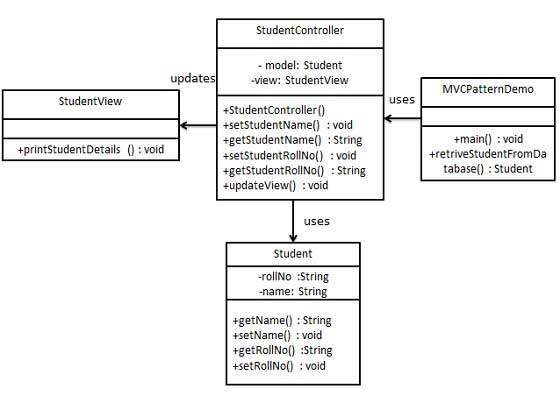
## Model View Controller pattern

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

**Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

**View** - View represents the visualization of the data that model contains.

**Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.



## Template pattern

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behavior pattern category.



# Implementation

Each class designed in this project has a specific task. We’ll continue by presenting the tasks for each Class.

The Dictionary interface provides the main functionality of the dictionary which is later going to be implemented in the Dictionary Class

**package** DictionaryPack;

**import** java.util.Map;

**import** java.util.Set;

/\*\*

\* **@author** Vlad

\*

\*/

**public** **interface** DictionaryProc {

/\*\*

\* **@pre** word != null && !synonyms.isEmpty()

\* **@post** dictionary.size() = previous dicitonary.size() +1

\* **@param** word

\* **@param** synonyms

\*/

**public** **void** **addWord**(**String** word,**String** synonyms,Map<String,Set<String>> dict);

/\*\* **@pre** word!=NULL and synonym!=NULL

\* **@post** new dictionary.get(word).size() == previous dictionary.get(word).size() + val;

\* **@param** 2 Strings : the word and the synonym to be added

\* **@return**

\*/

**public** **void** **addSynonyms**(**String** word, **String** synonym,Map<String,Set<String>> dict);

/\*\* **@pre** word!=NULL and synonym!=NULL

\* **@post** new dictionary.size() = oldDictionary.size() -1;

\* **@param** 2 Strings : the word and the synonym to be added

\* **@return**

\*/

**public** **void** **deleteWord**(**String** word);

/\*\*

\* **@pre** !dictionary.isEmpty()

\* **@post**

\* **@param**

\* **@return**

\*/

**public** **boolean** **isConsistent**(Map<String,Set<String>> dict);

}

The Dictionary class implements all the methods that are needed for a dictionary to be operational.

**package** DictionaryPack;

**import** java.util.HashSet;

**import** java.util.Map;

**import** java.util.Set;

**import** FileOperations.Serialization;

**import** FileOperations.SerializationJSON;

**import** FileOperations.SerializationSer;

**public** **class** **Dictionary** **implements** DictionaryProc {

**public** Map<String, Set<String>> dictJSON;

**public** Map<String, Set<String>> dictSer;

**private** **Serialization** sJSON;

**private** **Serialization** sSer;

**public** **Dictionary**() {

sJSON = **new** SerializationJSON();

sSer = **new** SerializationSer();

dictJSON = sJSON.readFromDictionary();

dictSer = sSer.readFromDictionary();

}

***@Override***

**public** **void** **addWord**(**String** word, **String** synonyms,Map<String,Set<String>> dict) {

**assert** word != **null** : "Word is null";

**assert** synonyms!=**null** : "Synonym is empty";

**int** **valPrecDict** = dict.size();

**if** (dict.containsKey(word)) {

addSynonyms(word, synonyms,dict);

} **else** {

Set<String> **setSyn** = **new** HashSet<String>();

setSyn.add(synonyms);

dict.put(word, setSyn);

**if** (!dict.containsKey(synonyms))

addWord(synonyms, word,dict);

**assert** dict.size() > valPrecDict: "Something went wrong";

}

**assert** isConsistent(dictJSON):".json dictionary is incosistent";

// assert isConsistent(dictSer):".ser dictionary is incosistent";

}

***@Override***

**public** **void** **addSynonyms**(**String** word, **String** synonym,Map<String,Set<String>> dict) {

**assert** word != **null** : "Word is null";

**assert** synonym != **null** : "Synonym is null";

**int** **valPrec** = 0;

valPrec = dict.get(word).size();

dict.get(word).add(synonym);

dict.put(word, dictJSON.get(word));

**if** (!dictJSON.containsKey(synonym))

addWord(synonym, word,dict);

**assert** dictJSON.get(word).size() >= valPrec : "Error adding the synonym";

**assert** isConsistent(dictJSON):".json dictionary is incosistent";

**assert** isConsistent(dictSer):".ser dictionary is incosistent";

}

***@Override***

**public** **void** **deleteWord**(**String** word) {

**assert** word != **null** : "Word is null";

dictJSON.remove(word);

**for** (**String** **s** : dictJSON.keySet()) {

**for** (**String** **s1** : dictJSON.get(s))

**if** (s1.equals(word)) {

dictJSON.get(s).remove(s1);

**break**;

}

}

**try** {

**for** (**String** **s** : dictJSON.keySet())

**if** (dictJSON.get(s).size() == 0)

deleteWord(s);

} **catch** (**Exception** **e**) {

}

**assert** isConsistent(dictJSON):".json dictionary is incosistent";

**assert** isConsistent(dictSer):".ser dictionary is incosistent";

}

***@Override***

**public** **boolean** **isConsistent**(Map<String,Set<String>> dict) {

**boolean** **exists** = **true**;

**if** (dict.size() != 0)

**for** (**String** **s** : dict.keySet()){

**if**(dict.get(s).size()==0){

exists = **false**;

}

**for** (**String** **s1** : dict.get(s))

exists = exists & dict.containsKey(s1);

}

**else**

exists = **false**;

**return** exists;

}

}

The super class which presents the main methods that are performed on a file:

**package** FileOperations;

**import** java.util.Map;

**import** java.util.Set;

**public** **abstract** **class** **Serialization** {

**protected** **String** filename = "Dictionary";

**public** **abstract** **void** **writeInDictionary**(Map<String,Set<String>> dictionary);

**public** **abstract** Map<String,Set<String>> **readFromDictionary**();

}

Next we are going to presents the implemented methods in one of the subclasses which extend the Serializable class.

**package** FileOperations;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.Iterator;

**import** java.util.Map;

**import** java.util.Set;

**import** java.util.TreeMap;

**import** java.util.TreeSet;

**import** org.json.simple.JSONArray;

**import** org.json.simple.JSONObject;

**import** org.json.simple.parser.JSONParser;

**public** **class** **SerializationJSON** **extends** **Serialization**{

***@SuppressWarnings***("unchecked")

***@Override***

**public** **void** **writeInDictionary**(Map<String, Set<String>> dictionary) {

**JSONObject** **dictionaryObj** = **new** JSONObject();

**for** (java.util.Map.Entry<String, Set<String>> **entry** : dictionary.entrySet()) {

**JSONArray** **jsonObj** = **new** JSONArray();

**for** (**String** **synonym** : entry.getValue()) {

jsonObj.add(synonym);

}

dictionaryObj.put(entry.getKey(), jsonObj);

}

**try** {

**FileWriter** **file** = **new** FileWriter(**super**.filename+".json");

file.write(dictionaryObj.toJSONString());

file.flush();

file.close();

} **catch** (**IOException** **e**) {

e.printStackTrace();

}

}

***@SuppressWarnings***("unchecked")

***@Override***

**public** Map<String, Set<String>> **readFromDictionary**() {

Map<String,Set<String>> **output** = **new** TreeMap<String,Set<String>>();

**JSONParser** **parser** = **new** JSONParser();

**try** {

**Object** **obj** = parser.parse(**new** FileReader(**super**.filename+".json"));

**JSONObject** **jsonObject** = (**JSONObject**) obj;

Set<String> **keys** = (Set<String>) jsonObject.keySet();

**for**(**String** **key** : keys){

Set<String> **values** = **new** TreeSet<String>();

**JSONArray** **arr** = (**JSONArray**) jsonObject.get(key);

Iterator<String> **iterator** = arr.iterator();

**while** (iterator.hasNext()) {

values.add(iterator.next());

}

output.put(key, values);

}

} **catch** (**FileNotFoundException** **e**) {

e.printStackTrace();

} **catch** (**IOException** **e**) {

e.printStackTrace();

} **catch** (org.json.simple.parser.**ParseException** **e**) {

e.printStackTrace();

}

**return** output;

}

}

# Results

The results can be seen in the JTables present in the graphic user interface. After entering the information in the text fields and pressing the add word , the information will be transmited and it will be written in the bank.ser file or in the .json file , if and only if the word does not already exist. Afterwards , the user could press the see synonyms for that specific word and it will show all the synonyms that are related to the word that was given as key.  
As another result we can explain what the search button does. Once the data in inputed in the text fields the user will select check the box he or she wants ( these boxes represent the dictionary ) and the program will search for each word that matches the criteria . The user is not bound to input the exact word, he or she can input only the first letter , the last or whichever letter followed by an “ \* “ and the program will display all the words which contain that letter , further more if the user does not know a letter an “ ? “ can replace that letter for example the user searches “?ast” the dictionary might print “fast” and “last” since they both contain “ast” .

# Conclusion and future development

Well , being the last homework for this semester I’ve spent a great deal of time trying to make it look and work as good as possible, but no one is perfect so my application might need a few patches. During the time I’ve spent researching I managed to learn about design patterns ( I realized I was using design patterns without even knowing it ) , improve my luck on guessing the regex and I improved my OOP skills. My application could use the following future development but I am not quite sure that this application is worth any more of my time , so here they are:

* Creating separate user interfaces for the administrator and the users so that the users won’t be able to add words but make a request to the admin to do it for them
* Adding more dictionaries available for the clients(different types, different languages,etc)
* Improving the efficiency of some algorithms
* Populate the dictionaries with a few thousand words
* Adding new fields for the gender of a noun or anything that would bring my eDictionary closer to a real dictionary