Programming Techniques

Assignment 5

Student : Rujita Alexandra

Group : 30422

Contents

[1. Problem specification. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3](#_Toc448226874)

[2. Problem analysis , modeling, scenarios , use cases. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3](#_Toc448226875)

[2.1. Problem analysis 3](#_Toc448226876)

[2.2. Modeling 4](#_Toc448226877)

[2.3. Use cases 4](#_Toc448226878)

[3. Design. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_Toc448226879)

[3.1. UML Class Diagram 5](#_Toc448226880)

[3.1.1. CashRegister Class 6](#_Toc448226881)

[3.1.2. FileOp Class 6](#_Toc448226882)

[3.1.3. Generator Class 6](#_Toc448226883)

[3.1.4. GUI Class 6](#_Toc448226884)

[3.1.5. Server Class 6](#_Toc448226885)

[3.1.6. Task Class 7](#_Toc448226886)

[3.1.7. mainC Class 7](#_Toc448226887)

[3.2. Data structures 7](#_Toc448226888)

[3.3. Arhitecture 7](#_Toc448226889)

[3.4. Relationships 7](#_Toc448226890)

[3.5. Algorithms 7](#_Toc448226891)

[3.6. User interface 8](#_Toc448226892)

[4. Implementation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_Toc448226893)

[5. Further developments and importance. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11](#_Toc448226894)

[6. References. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc448226895)

1. Problem specification

Consider the implementation of one of the following:

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

1. Problem analysis , modeling , scenarios , use cases

## Problem analysis

As specified in the problem’s specification this requierment can be done in 2 ways: either as a synonyms dictionary or as a language dictionary. Either way, all the words that make up the definition of the principal word must also appear in the dictioary with their own definition or synonyms.

When implementing the program you have to consider also the user and the admin, where the user can only search for a word or for a word’s synonyms / definition and the admin can add and remove words or add new synonyms to an already existing word. There is also the opton of seein the hole dictionary

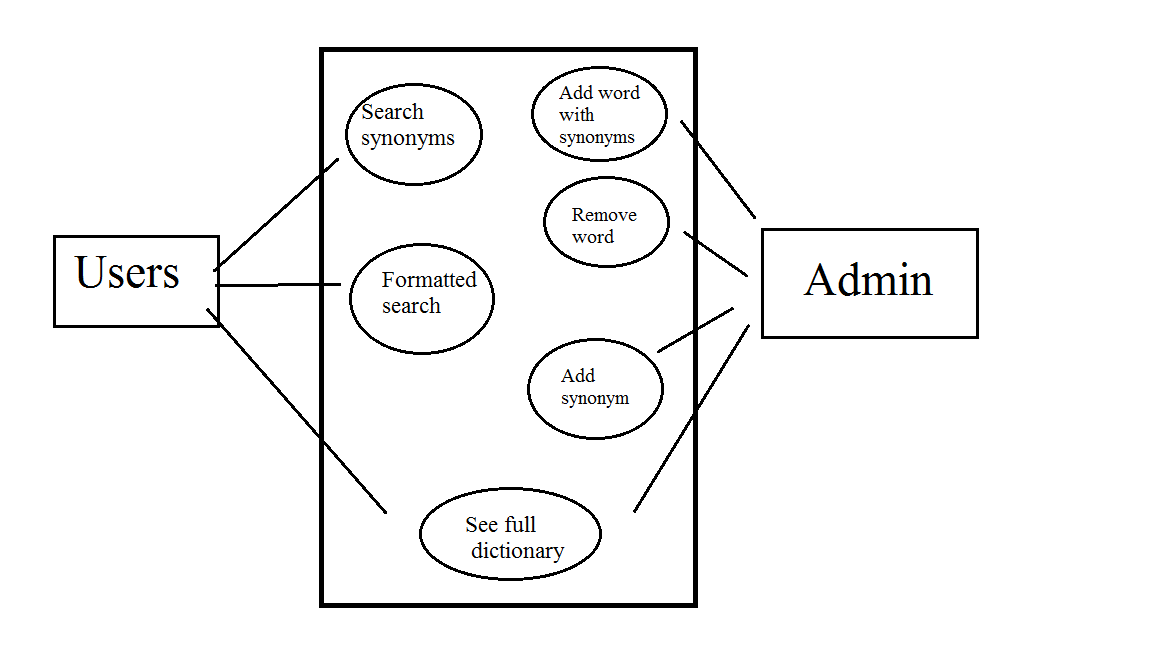
## Modeling

In this implementation I choose to read data from a serializable JSON jackson text file every time I run the program, to retrieve the information and populate the dictionary’s records. Also, everytime I add a new word and it’s synonyms, remove it, or edit it the file is rewriten with the new information from the dictionary’s records

Only the admin can add or remove words, but the user can also add a new synonym for an already exiting word.

The user can search for a word(s) using the \* or the ? sing, if he does not knwo exactly what he is searching for or if he wants the word to contain specific letters.

## Use cases

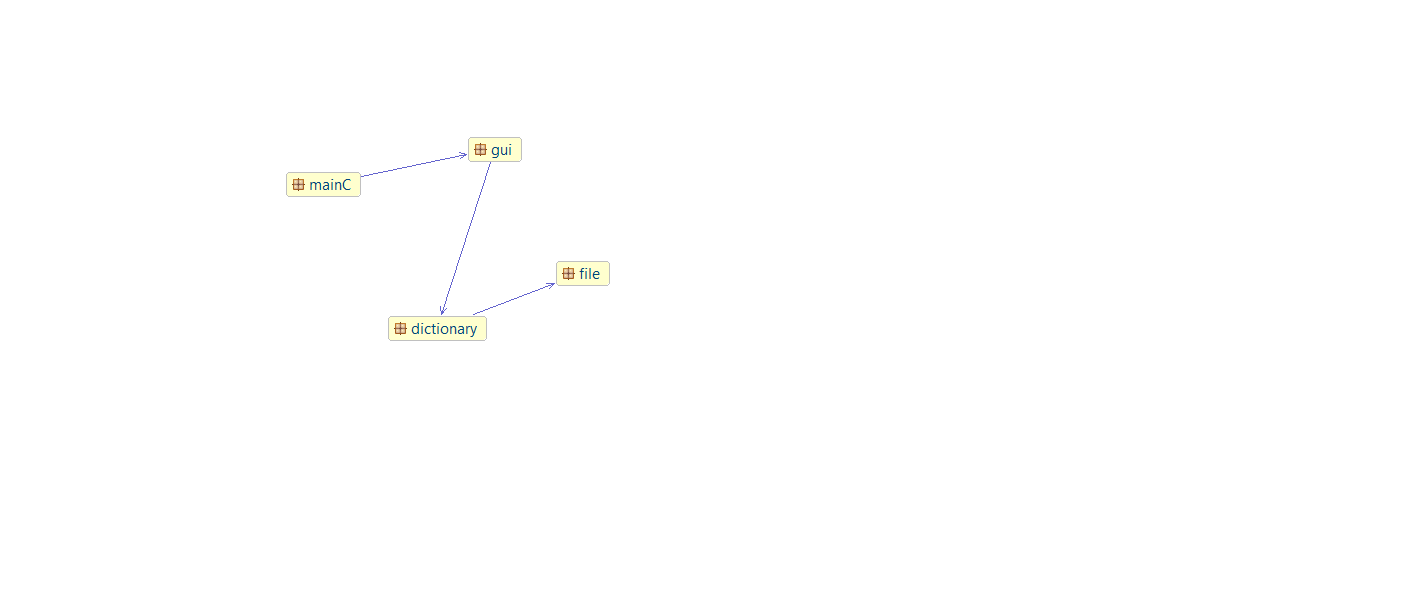


As seen in the picture above, users can search for the synonyms of a word or for words that match their own criteria. Admins can add or remove a word, with their synonyms as well and can add a synonym for an already existing word. Both the user and the admin can see the full dictionary by a simple button press.

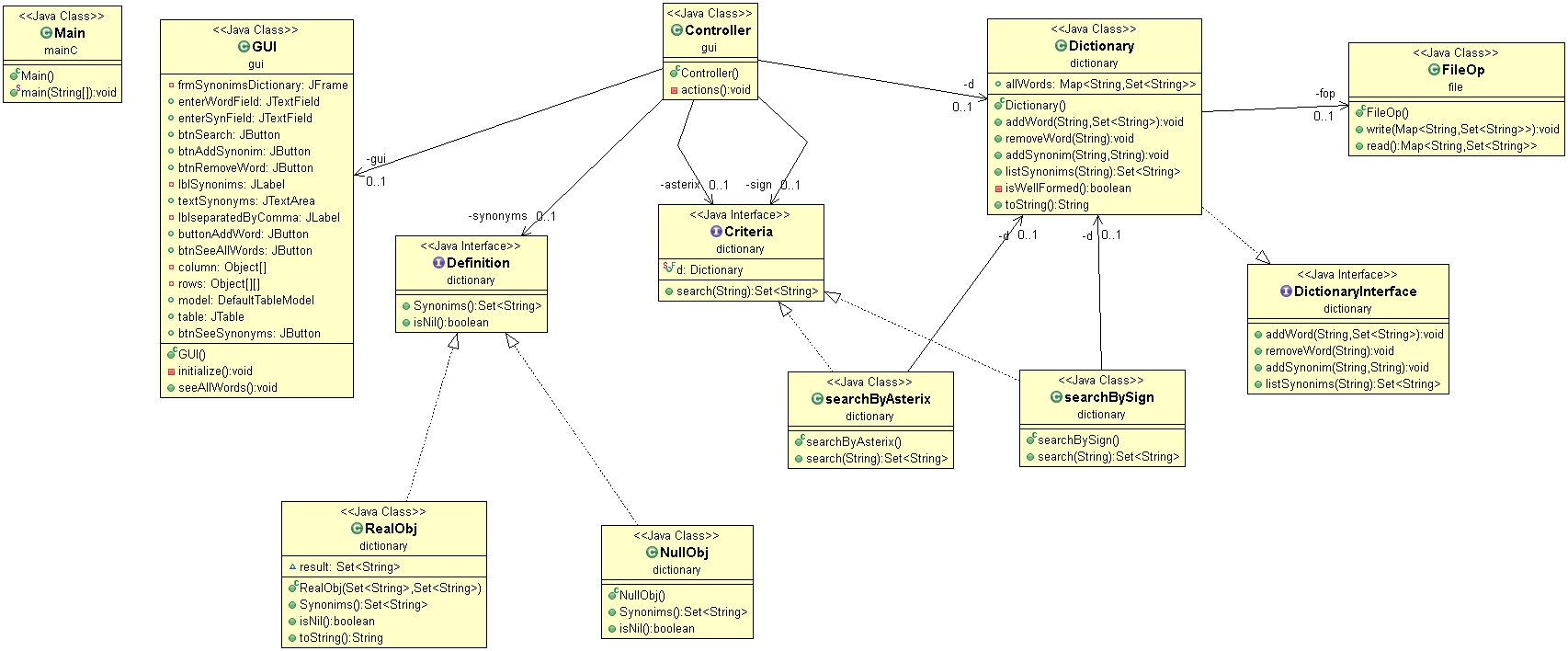
1. Design

## Package diagram

Package Dependencies Diagram shows the dependencies between packages and highlights circular dependencies. Each dependency line also displays the involved classes.



## UML Class Diagram

****The UML diagram is a class diagram that shows a set of classes, interfaces and collaborations and their relationships. Class diagrams may also contain packages or sub-systems, both of which are used to group elements of the model.

### DictionaryInterface Interface

This interface implements the design by contract pattern. It holds the headers of the methods and also all the pre and post conditions that must be fullfiled by the execution of those methods.

### Dictionary Class

This class implements the DictionaryInterface Interface and all it’s methods and also some new one, which are requiered for the printing of the data. Also all the methods implement the pre and post conditions through the assert statement. There is also a isWellFormed method which checks if the dictionary si consistent or not, before and after the execution of each method.

The main data structure that holds all the data is a hashMap having as key a String and as value a set of Strings.

### FileOp Class

In this class we perfom all the operations on file. For this assignment the JSON jackson serialization and deserialization of file is used, making the application’s data portable to other programming languages.

### Criteria Interface

This interface is used to implement the Filter design pattern. It holds just a method which returns the set of Strings that match the criteria.

### searchByAsterix Class

In this class, that implements the Criteria interface the search method is implemented. This method accepts the passed string only if it contains an asterix,which can replace an entire tring ( sequence of charcters) or a null string. If so it transforms the string into a regex and by passing through all the dictionary records it checks which word matches the regex. When a word is matched it is added into a set of strings, which is the result of the called method.

### searchBySign Class

In this class, that implements the Criteria interface the search method is implemented. This method accepts the passed string only if it contains an question mark, which replaces a single letter. If so it transforms the string into a regex and by passing through all the dictionary records it checks which word matches the regex. When a word is matched it is added into a set of strings, which is the result of the called method.

### Definition Interface

This interface is used to implement the nullObj design pattern. It has the isNill method and another method which will be the actually result.

### RealObj Class

This class gets in its’ constructor as parameters 2 sets of strings and goes through each one, and in a resulting set adds all the common elements from both sets ( it does an intersection of the sets). Then this set is returned by the synonyms method.

### NullObj Class

In this class it is treated the null case, when the dictionary has nothing to print as result it will print just a message „Word is not in the dictionary yet”.

### 3.2.10 Controller Class

In this class all the action listener’s for the buttons are written. Also here all the printing in the table is done and the processing of strings, such as the input string that contains several synonyms separated by a comma and a space.

Also here after implementing the filter pattern the RealObj constructor is called to do the intersection of the 2 resulted sets. And if both of them have the size equal to zero then the nullObj class will be called.

### 3.2.11 GUI Class

In this class only the definition of the buttons and everything that make up the user interface is done, as well as the constructing a new frame for displaying the table of all the words and their synonyms in the dictionary.

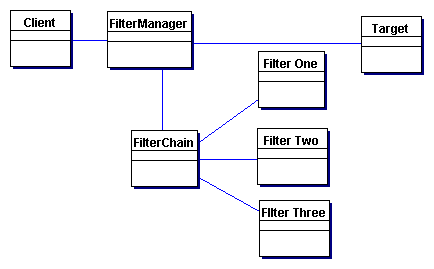
### 3.2.12 mainC Class

This class is the main class of the project that makes a simple Controller object so that the user interface and all its’ functionality is shown.

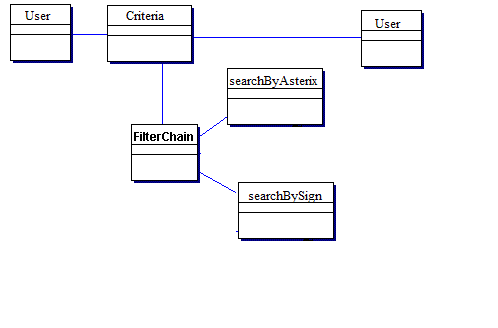
## Data structures

The main data structure used here is the HashMap that has as key a String and as value a set of Strings. All the data is stored in a JSON jackson file.

## Design patterns

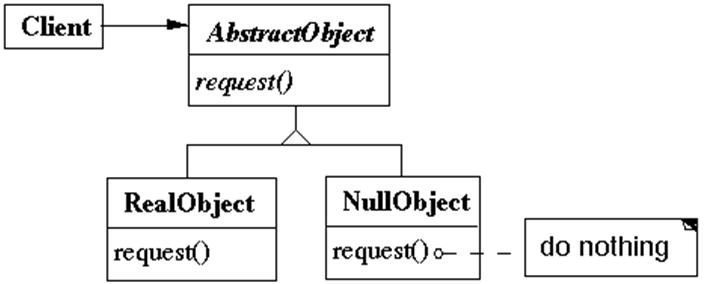
First of the design patterns that I used is the Filter design pattern. Filter pattern or Criteria pattern is a design pattern that enables developers to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations.

## 

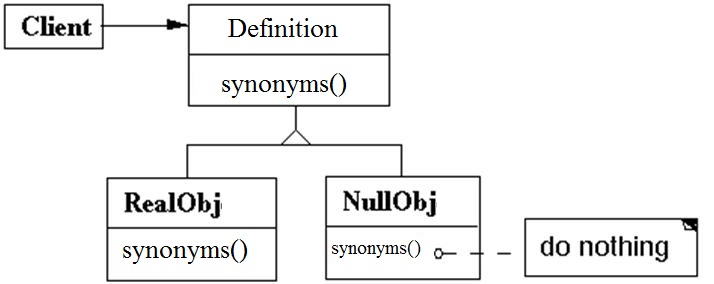
The design pattern diagram for my project is the next one:

The second design pattern that I used is the Null Object one. In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do nothing relationship. Such Null object can also be used to provide default behavior in case data is not available.

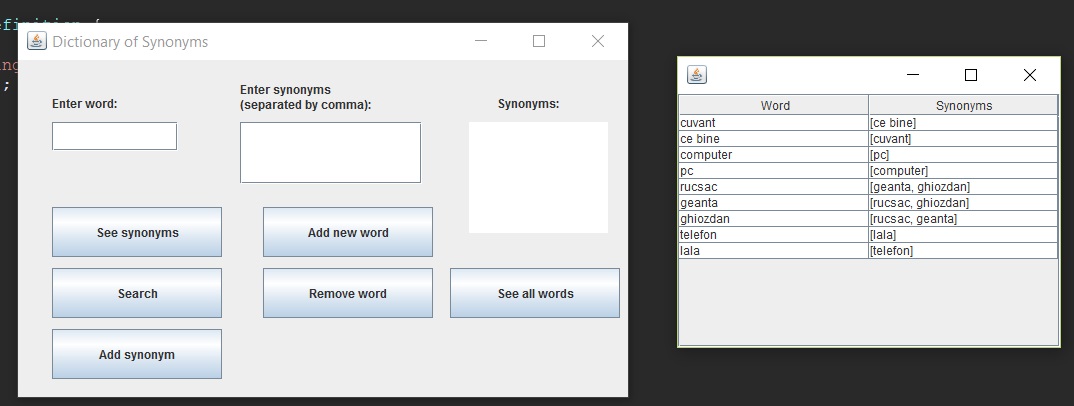
In Null Object pattern, we create an abstract class specifying various operations to be done, concrete classes extending this class and a null object class providing do nothing implementation of this class and will be used seemlessly where we need to check null value.



Applied for my application is the following diagram:

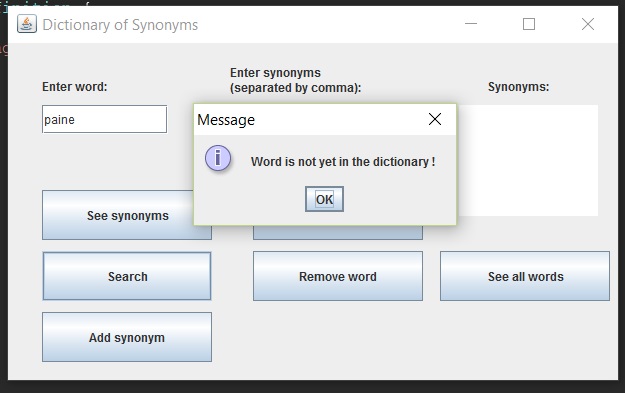


## User interface



As seen from the user interface there are several operations that can be done.

When entering a new word with a list of synonyms it is specified that they should be separated by a comma. When pressing the „ See all words „ button a new frame that contains all the words and the associated synonyms. Also it can be noted that all the words taht appear on the left hand side also appear on the right hand side.



When searching for a word that is not yet in the dictionary an information message appears informing the user that the word is not yet added to the dictionary. This is done by calling a NullObj.

1. Implementation

The method search() from the Criteria interface implemented in both of the classes:

**public** Set<String> **search**(**String** word) {

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

**if** (word.indexOf('\*') >= 0) {

word = word.replaceAll("\\\*","(.\*)");

**Pattern** **p** = **Pattern**.*compile*(word);

**Matcher** **m**;

**for** (**String** **toAdd** : d.allWords.keySet()) {

m = p.matcher(toAdd);

**if** (m.lookingAt()) {

result.add(toAdd);

}

}

}

**return** result;

} ***@Override***

**public** Set<String> **search**(**String** word) {

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

**if**(word.indexOf('?') >= 0){

word = word.replaceAll("\\?", "[^ ]");

**Pattern** **p** = **Pattern**.*compile*(word);

**Matcher** **m**;

**for** (**String** **toAdd** : d.allWords.keySet()) {

m = p.matcher(toAdd);

**if** (m.lookingAt()) {

result.add(toAdd);

}

}

}

**return** result;

Also the constructor of the RealObj and the implementation of the synonyms() method:

**public** **RealObj**(Set<String> s1, Set<String> s2) {

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

**if** (s1.size() == 0 && s2.size() != 0)

result.addAll(s2);

**else** {

**if** (s2.size() == 0 && s1.size() != 0)

result.addAll(s1);

**else**

**for** (**String** **s** : s1) {

**if** (s2.contains(s))

result.add(s);

}

}

**this**.result = result;

}

***@Override***

**public** Set<String> **synonyms**() {

// **TODO** Auto-generated method stub

**return** result;

And in the NullObj class the synonyms() method is implemented as it follows:

***@Override***

**public** Set<String> **synonyms**() {

**JOptionPane**.*showMessageDialog*(**null**, "Word is not yet in the dictionary !");

**return** **null**;

In the dictionary class the pre and post conditions are implemented as it follows:

/\*

\* @Pre word does not exists && synonims != null

\* @Post preNrWords + 1 = postNrWords && newSynonims are defined

\*/

**void** **addWord**(**String** w, Set<String> synonims);

***@Override***

**public** **void** **addWord**(**String** w, Set<String> synonims) {

**assert** isWellFormed() : "Dictionary not consistent";

**assert** !w.equals(**null**) : "Word is null";

**int** **preSize** = allWords.size();

**int** **postSize**;

allWords.put(w, synonims);

**for** (**String** **s** : synonims) {

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

newSyn.add(w);

**for** (**String** **newS** : synonims) {

**if** (!s.equals(newS))

newSyn.add(newS);

}

allWords.put(s, newSyn);

}

postSize = allWords.size();

fop.write(allWords);

**assert** postSize > preSize : "Words not added";

**assert** isWellFormed() : "Dictionary not consistent";

}

}

1. Further developments and importance

As further developments more words should be added to the database and also maybe some sort of reference to a definition of that word.

As for importance thanks to this application I learnt how to use the JSON Jackson file which are cross platform, a very important thing nowadays when you have to work with not just one platform all over the wolrd.