**Technical University of Cluj-Napoca**

**Faculty of Automation and Computer Science**

**Computer Science Department**

**Programming Techniques**

**Fifth Homework: Dictionary**

**Mihoc Nicolae**

**Group: 30425**

Table of Contents

*1. Problem Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3*

*1.1 Problem Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3*

*2. Diagram Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*

*2.1 Use Case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4*

*2.2 Class Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5*

*2.3 Sequence Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6*

*2.4 Package Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*3. Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*3.1 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*

*3.2 Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*

*4. Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9*

*5. Improvements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12*

*6. Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12*

*7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13*

**1. Problem Specification**

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.

**1.1 Analysis of the Problem**

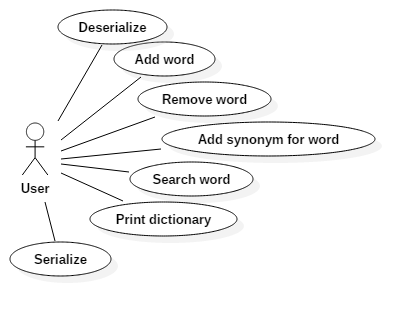
We are required to implement a Dictionary application. The available options are: romanian or english dictionary or a romanian or english dictionary with synonyms. I chose the romanian dictionary with synonyms. In order to implement this homework we are required to use Java Map collection so I chose the TreeMap collection. The user must be able to add words, remove words, populate the dictionary and search for a word. At the searching part there is a trick and that is to be able to search for a word with a missing character or for a part of the word and to show all possible options. The final thing is about the consistency of the dictionary. Each time we add a word the synonym must also be added to form the consistency.

**2. Diagram Implementation**

For the purpose of better understanding how to resolve this problem first we need to draw the neccessary diagrams. These diagrams will help us to understand how to approach the final solution. We need multiple diagrams because each one shows a different angle related to the problem.

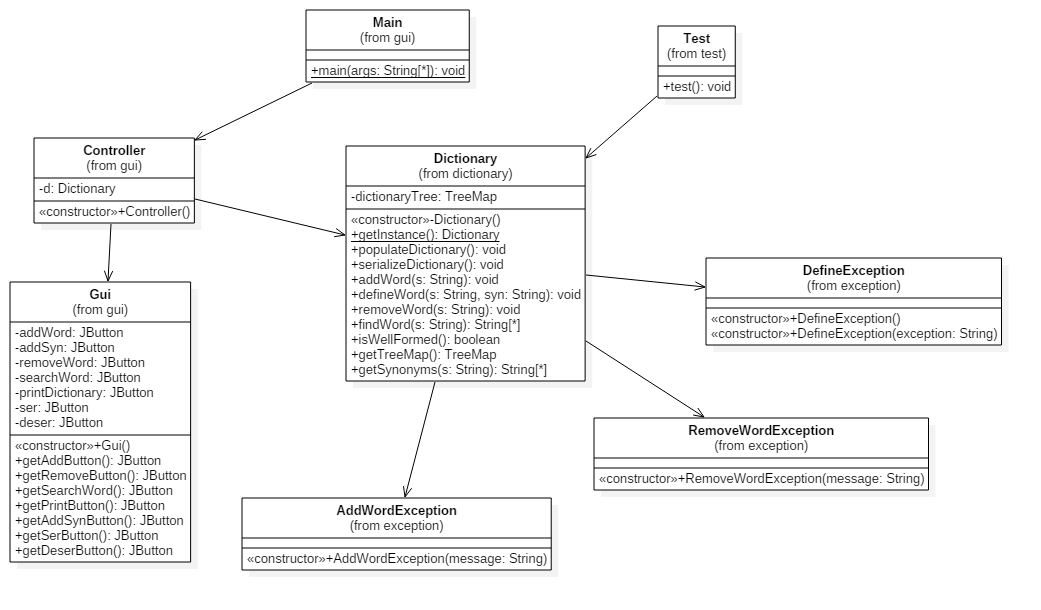
**2.1 Use Case Diagram**

The first diagram is the use case diagram. This diagram is the simplest one because it shows only the available options of the application for the user and it doesn’t approach any details of the implementation. It is useful beacause it shows the final possible actions that can be used by the user.



**2.2 Class Diagram**

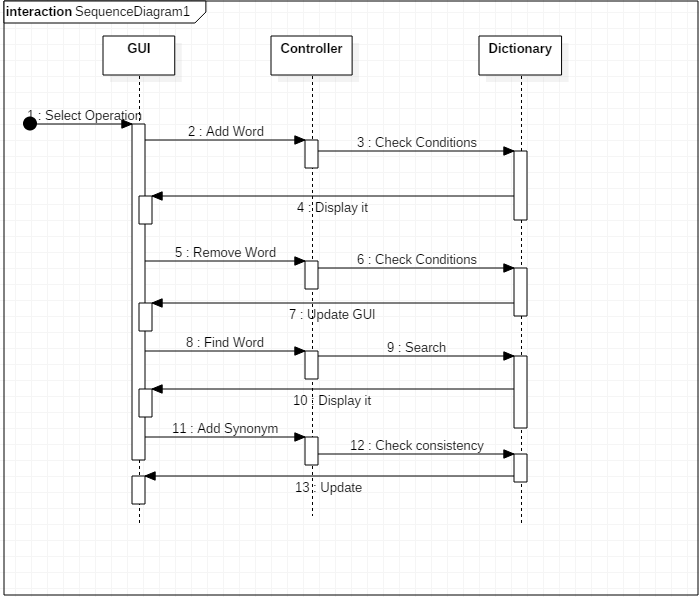
The second diagram is the class diagram. It is the most important diagram and the most complex one because it captures the attributes and the methods of every class. Another important thing about this type of diagram is that it shows also the connection made between this classes like aggregation relations, composition relations etc.



The main attraction of this class diagram is the Dictionary class which does all the magic. All the methods implemented there come from the interface DictionaryInterface and they are accessed by the controller in the graphical user interface. Also the Dictionary class is able to check and throw exceptions if they match the implemented ones.

**2.3 Sequence Diagram**

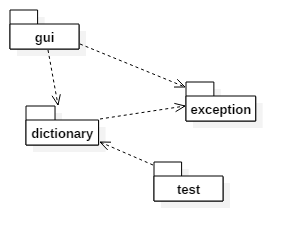
This diagram has the purpose of showing what happens exactly when we give the application a command. It displays the whole process which happens in the background based on what we coded starting from the moment we give the command to the moment the changes are displayed in the graphical user interface.



The order the operations are chosen is irrelevant. They are controlled by the controller class and executed if the conditions met by the Dictionary class which returns the result back to the graphical user interface to display them.

**2.4 Package Diagram**

The package diagram is very simple for this implementation because there are not many packages lying around in the project. The connection between them is based on the usage relations between them.



The “gui” package points to the “dictionary” package because it displays its contents, points to the “exception” package because it checks for them and also the “dictionary” package points to the exceptions because it uses them.

**3. Design**

After the presentation of the diagrams it is time to think about a strategy to design our application in code. It is important to decide the minimum and enough number of classes that we need and also the packages to store these classes.

**3.1 Packages**

For this implementation I decided that 4 packages should suffice. I split the classes in the following packages:

-gui: this package contains 3 classes and these are: Gui which creates the graphical user interface, Controller which handles the behaviour of each button pressed accordingly and the Main class which makes this all work.

-exception: this package contains 3 classes also and these are: AddWordException, DefineWordException and RemoveWordException. These classes handle the events when exceptions are encountered.

-test: this package has only the JunitTest class test which tests the functionality of the implementation.

-dictionary: this is the main package in terms of importance because it contains the 2 most important classes of this implementation and these are : Dictionary and DictionaryInterface which handle the whole logic of the application.

**3.2 Classes**

The total number of classes that I considered to be neccesary for this implementation is 8. In order to develop this dictionary application I needed a few things. First was the creation of the “DictionaryInterface” interface which will contain all the neccessary methods the “Dictionary” class will implement like the addition of a word, removal of a word, finding a word or serialization or the method to populate the dictionary. All these must have some pre and post condtitions attached as comments but implemented in the Dictionary class using assert.

Also this project must implement 2 design patterns. I chose the “Singleton” design pattern and the MVC(Model-View-Control) patern. I will explain the Singleton pattern when I will describe the Dictionary class. About the Model-View-Control design pattern I can say that it is used in this implementation under the following form: the model is represented by the Dictionary class which will develop the methods neccesary to build a real dictionary, the controller will be represented by the “Controller” class which handles the action listener of the buttons. All the methods invoked by this Controller class must respect the model of the Dictionary class and finally the View part which is represented by the graphical user interface class which displays the result of any method invoked earlier which respects the proposed model.

**4. Implementation**

After we disscused the classes and packages that we need to get the job done we must talk about the actual implementation of all this.

*Dictionary Interface*

In this interface are declared the main methods used in this application which must be implemented by the Dictionary class. The methods that this interface contains are the following: a method to serialize, a method to deserialize, a method to addWord, removeWord, defineWord or findWord. Each of these methods have pre and post condtions. Let’s take an example: the method addWord takes as parameter a string throws a AddWordException if the word was already added and has the follosing pre conditions: what we enter must be a string, musn’t be null and as a post conditions: the method isWellFormed must be true, that means that the consistency must be checked after that.

*Dictionary Class*

The most important class in this whole project is this class which handles the logic of the dictionary. The methods metioned in the “Dictionary Interface” are implemented here. This class implements that interface and also the Serializable interface . The first important thing that needs to be mentioned here is the Singleton design pattern which is implemented in the following way: a private static Dicitonary item which is instantiated with null is created, the constructor of this class is declared protected because it musn’t be accesed from another place and after that the method getInstance is defined which gives the Dictionary item the value returned by the constructor. Basically the Singleton design pattern states that the model cannot be instantiated more than one time and must be from within the class and not from another place.

Coming back to the implementation of this class I chose the TreeMap implementation which takes as parameters a string and an arraylist of strings. The TreeMap java collection stores the value at the right key, in this case the synonym to its rightfull word.

First method is called populateDictionary() and is the deserialization process from a created file which allows us to see all the words entered on the screen. Secon method is the serializeDictionary() which saves all the words entered after so we can have them after the application is closed.

After those 2 methods come the most important ones. The addWord(String s) method gets a String as parameter, throws the AddWordException if the word already exists and first checks for the pre conditions implemented with assert if they are true a new word is added in the TreeMap on the right place. The defineWord(String s, String Syn) method also checks for the pre conditions, throws the DefineException and makes sure the defined word and synonym are both implemented on the left side and right side of the dictionary at once to be sure the consistency is true.

*AddWordException Class*

A class that extends Exception and calls in the constructor the message of the exception.

*DefineWordException Class*

A class that extends Exception and calls in the constructor the message of the exception.

*RemoveWordException Class*

A class that extends Exception and calls in the constructor the message of the exception.

*Controller Class*

This is the class that has the controll over all available methods. Every method is described here by a Jbutton from the graphical user interface and each button extends the ActionListener method.

The first thing that this class does is to create an object of type Dictionary to use in the following methods and to call a new graphical user interface object.

The Jbutton “deserButton” calls the method populateDictionary() which means it deserializes the data.

The Jbutton “addbutton” adds the given word to the dictionary and prints the exception if necessary.

The Jbutton “searchWord” checks in the dictionary for the coressponding word and creates a new frame where it displays the searched word if any is found.

The Jbutton “printButton” is responsible with printing the data that was deserialized and place it in a Jtable with the words on the left and the synonyms on the right.

The Jbutton “addSynButton” for a given word it adds the coressponding synonym in the dictionary.

The Jbutton “removeButton” removes the given word from the dictionary if it exists or throws the exception if not.

The Jbutton “serButton” performs the serialization of the data once we are done with all the other methods and we want to save our changes after the application is closed.

*GUI Class*

This class is responsible with the design of our application. In my case I opted for a simple interface which contains only the buttons , one for each method from the dictionary class which are placed in a GridLayout. The changes are displayed in a different window and the data is entered in a JoptionPane text area. In this class the only attributes that are declared are the Jbuttons and in the constructor the frame is initialized and a panel is added. In that panel the layout is set to GridLayout and the buttons are added to it. Also this constructor contains the usual things like visibility, position, size of the window and the DefaultCloseOperation option.

Besides the constructor there are the get methods for each button which is used in the controller class.

*Main class*

The final class is the Main class whose only job is to call for a new Controller() every time the application is launched. Without this the program would never work.

*Test class*

For this application I also implemented a Junit test class whose job is to check if the methods that I implemented work properly and if they do a success message will be shown. I tested the addWord method by adding a word to the dictionary and after the defineWord method which will add the synonyms for that word as independent words. The final method that I tested was the removeWord method to remove the selected word and the coressponding synonym to preserve the consistency of the dictionary.

**5. Improvements**

This dictionary application that I implemented resambles really good a real dictionary and is even easier to use because of the consistency which defines all the words that are introduced. One thing that needs to be mentioned is the fact that my application is not listing words from a Json file or a XML file as required and that is one of the things that could be improved by introducing that.

**6. Conclusions**

This homework made me better understand the design patterns and futher more the pre conditions and post conditions. I gave a shot and tried to use the Json file type but with no success so I came back to a normal file. Also it showed me how to search for a word if I replace a character or more from a given word. The interface is simple and easy to use by anyone.

**7. Bibliography**

**1.** <http://stackoverflow.com/>

**2.http://www.tutorialspoint.com/design\_pattern/mvc\_pattern.htm/**

**3.http://www.tutorialspoint.com/design\_pattern/singleton\_pattern.htm/**

**. . . . . . . .. . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**