*Technical University of Cluj-Napoca*

*Faculty of Automation and Computers*

*Department of Computers*

*2nd Year, 2nd Semester 2015-2016*

Programming Techniques

Homework 1

Student: Robert Văcăreanu

Group: 30424

CONTENTS

1. Project specification
2. Problem analysis, model, testing and usage
3. Design
4. Implementation and testing
5. Results
6. Conclusions
7. Bibliography
8. Project Specification

TP Lab – Homework 5

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. Problem Analysis, Model, Testing and Usage
2. Problem Analysis

For this problem I’ve chose to implement a synonym (thesaurus) dictionary. Operations supported are: add, remove, search. Saving and loading is done with the help of a 3rd party library to json. The 2 design patterns implemented are: Memento and Commander.

Memento was chosen because we want to have a functionality like undo/redo when searching. Commander is used to interact with the dictionary.

1. Model

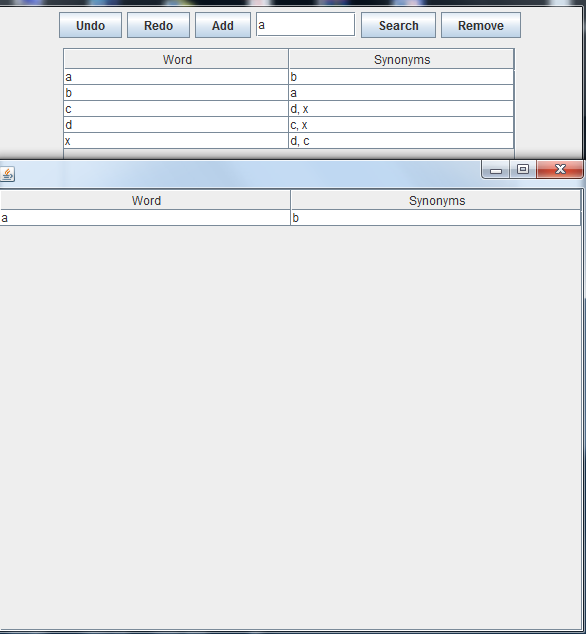
In order to model a dictionary, I have chosen an approach similar to composite design pattern. There is a class named Dictionary Entry. This is what you see in a typical dictionary: the word with its explanation. This class further contain a collection -- set, to be more specific, of words. This is the definition of the word. The class word is a wrapper around string. This is done to assure easily added features.

1. Testing

The correct functionality is assured by techniques from design by contract paradigm. Namely: assertion, well formed method and invariants. It is guaranteed that any argument will not be null, as well as synonyms are synonyms for each others. Furthermore, it is always true that in the definition of a word will appear only words that are already in the dictionary.

1. Usage

The end-user is free to: add words, delete words as well as search words.



The user interface, as seen in the picture above, is fairly easy to use. The button’s names are intuitive. Their operates as the user will anticipate. Namely:

By clicking “Add” an option pane will pop out asking for the word and it’s synonyms in order to add them in the dictionary. There is ok if you don’t have in the dictionary the synonyms you are trying to add since they will be automatically added.

After entering the word in the text field, you can either delete it or search it.

If you delete it, it will be removed from every entry. If you search, a new frame will pop out showing the results. For this implantation, I have used some lambda expressions together with streams.

The model contains 6 classes: AddWordObserver, Caretaker, DictonaryEntry, Memento, Originator and Word. Two interfaces: I\_DictionaryEntry, I\_Memorable.

The class AddWordObserver is used in order to notify the manager ( controller ) that there was a button pressed. Further. The controller delegates the according command using a hash map. Here comes into play the command design pattern.

The classes Caretaker, Memento and Originator are part of the Memento design pattern, together with the I\_Memorable interface. They are implemented in a generic way, so that they can be used to memorize not only words. This assures that it will be easy extended in the case that is needed more functionality.

Memento class is a wrapper around what we will memorize. Originator class represents the state of the object we want to memorize at a given time. The Caretaker contains a list of mementos.

The Dictionary Entry class contains a Word and a set of words. It is the row inside a dictionary.

The Word class is a wrapper around the string class

The interface I \_ Dictionary Entry defines the methods every dictionary entry must implement: get word, add word, remove word and get synonyms. This are the basic operations that you must be able to perform over a dictionary entry.

The interface I \_ Memorable makes the design pattern generic. It takes a generic parameter T and have two methods: restore and store to and from memento

The package controller contains 2 classes: Manager and Dictionary.

The Manager class is used to control the functionality at a higher level. In order to achieve this it uses: Main Window : a view, Dictionary : the actual dictionary and a hash map that links a string to a command. Every time an actions is triggered inside the view, by using the observer design pattern it sends a string to the observers. The string is automatically interpreted by the hash map. The values of the hash map are commands. You can dynamically add commands by the method provided. It takes as parameters a string and a Command

The Dictionary class is the representation at a higher level of a dictionary. It is modified only by the use of commands.

The package commands contains 5 classes: Add Entry, Commander, Macro, Remove Entry, Search Word and two interfaces: Command and I \_ Dictionary.

The interface I \_ Dictionary specifies the needed methods for a dictionary controller: add entry, remove entry and search word. It is implemented by the Commander class.

The interface Command have only one method : execute. Hence it is a functional interface

The class Commander contains the whole functionality. Its constructor is protected, hence only in this package we can instantiate it. It is aggregated by every command class. It contains functionality for operations such as : add entry to dictionary, remove entry from dictionary as well as search word through dictionary. Contains a method used for checking whether the data is compromised or not. Namely : well formed that returns a Boolean.

The classes Add Entry, Remove Entry and Search Word uses the commander class in order to provide functionality outside the package.

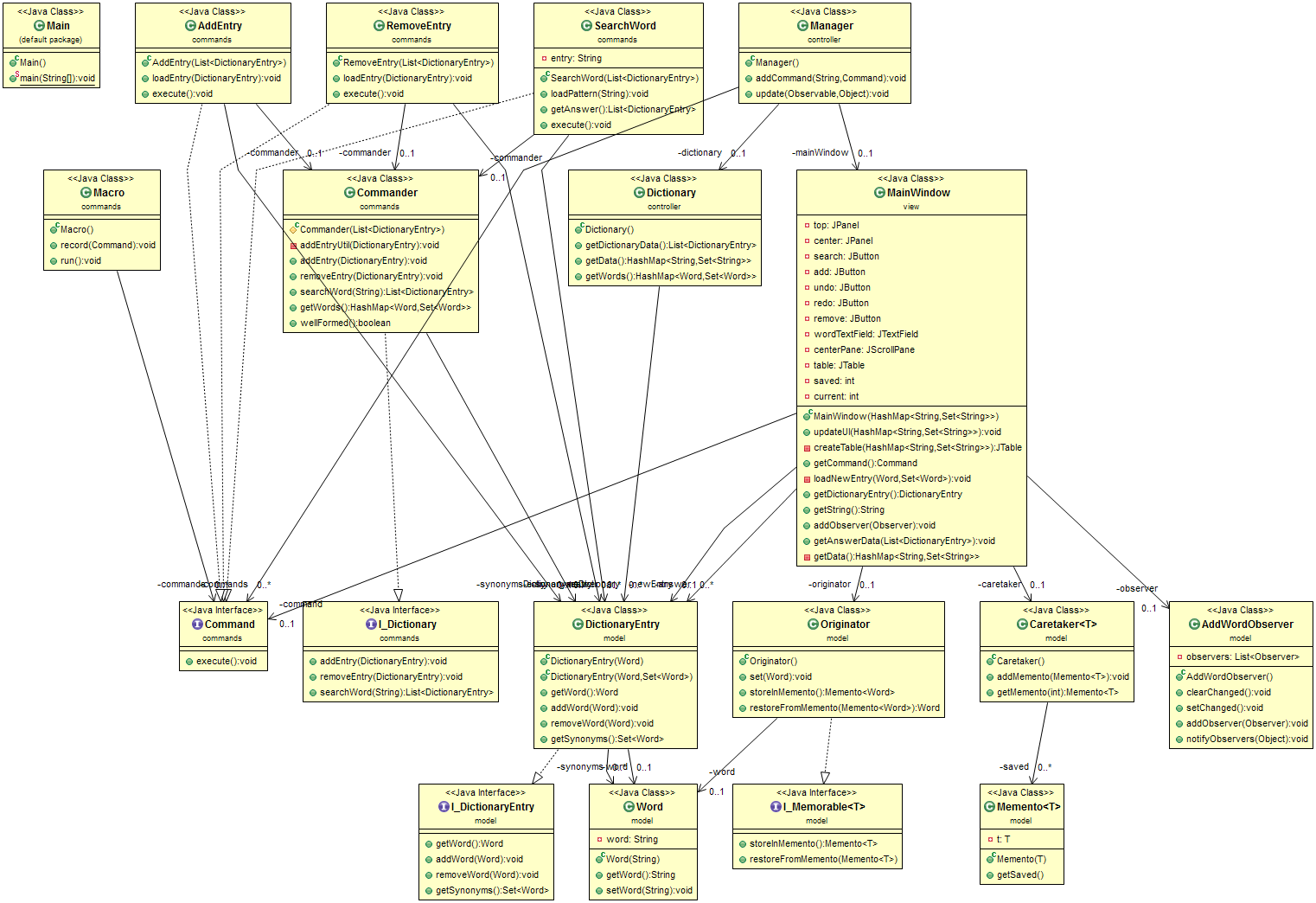
The package view contains only one class : Main Window. This class is responsible with providing the view to the user by the use of swing components.

It also uses classes used for implementing the Memento design pattern.

It is aggregated by the manager class and controlled using the command design pattern. It uses as layout manager the default one for frame : the border layout. The functionality for the components was implemented using anonymous inner classes that sets the action listener for every component that provides functionality for the user. Inside the method of the action listener it is called observer . notify Observers. It sends as argument a string that uniquely identifies an action to the manager.

1. Implementation and testing

The implementation was done according to the model – view – controller architectural design pattern guidelines.

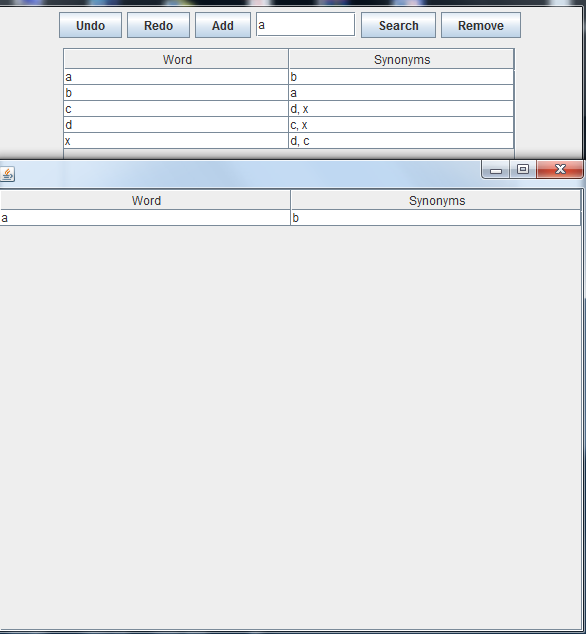


The testing was done on the go by using assertion, design by contract, invariants and the method called well formed. This assures that the consistency of the dictionary is not affected while performing operations on it.

As well as manually checking the outputs since this is a small project. However, in a bigger or more complex project that is prone to frequent modification the junit method of testing is the one to use since a modification in a class could affect more classes (chain) and testing them again would be time wasting. By writing test units, everything is automated and we only take a look at those classes if they failed the tests. Testing is very important and test units provide a way to efficiently manage it.

1. Results

We achieved the final product:



1. Conclusions

While the difficulty depends on the model and the way the programmer chose to implement it, I evaluate the difficulty as being somewhere between easy-medium. Firstly, because we have to take care of the specifications and to find an abstract way of modeling a dictionary instead of using plain old approach : String with list of strings. Also, the project involves user input that must be validated. This can become very hard since there are more rules for assuring the integrity of the dictionary.

I learnt from implementing this project how difficult can be to verify and modify the user input while giving him the freedom of putting the input however seems clear to him while also making sense to others. Also, I got the change of implementing some very nice design patterns that proved to be very useful.