**Assignment 5**

**Thesaurus**

Student: Nagy Hanna

Group: 30421

1. **Objectives of the problem**

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.

From the options above, I chose to implement Thesaurus and not Dictionary. There is no difference between their implementation and structure, the only difference is in the semantic field, but that is not aim of this homework.

My version of implementation of the assignment consists of the operations on the Thesaurus. For example, there is a possibility of adding a new word in the database, which is a XML file; one can also delete a certain word, or a certain synonym of a defined word; there is also a chance to search for a word with the regex (\* for any string or ? for one character). Of course, after these operations the database (XML file) is updated, saved and loaded again in the program.

1. Analysing, Modelling, Scenes, Class Diagrams

An important aim of this assignment was to use different design patterns through the implementation. I chose to use the Singleton design pattern and the MVC design pattern which stands for the M Model, V View and C Controller.

About the Singleton design pattern:

Sometimes it's appropriate to have exactly one instance of a class: window managers, print spoolers, and filesystems are prototypical examples. Typically, those types of objects—known as singletons—are accessed by disparate objects throughout a software system, and therefore require a global point of access. Of course, just when you're certain you will never need more than one instance, it's a good bet you'll change your mind.

The Singleton design pattern addresses all of these concerns. With the Singleton design pattern you can:

* Ensure that only one instance of a class is created
* Provide a global point of access to the object
* Allow multiple instances in the future without affecting a singleton class's clients

Although the Singleton design pattern—as evidenced below by the figure below—is one of the simplest design patterns, it presents a number of pitfalls for the unwary Java developer. This article discusses the Singleton design pattern and addresses those pitfalls.

For these characteristics I have chosen this design pattern. In my implementation of the homework the singleton class is the class named Thesaurus ( actually SingletonThesaurus). It could be easily understanded why it is needed for this class to be singleton one. So, in our implementation, this class is Singleton because just one instance of the class it is needed, we have one thesaurus book and we operate on the data which in in this thesaurus book.

This pattern it was implemented with the tricks of making the constructor private, having a private static variable of its own class to ensure that only one instance it is created and also we have a synchronized method which makes this ensurence described above.

**public** **static** **synchronized** SingletonThesaurus getInstance ( ) {

**if** (*thesaurus* == **null**) {

*thesaurus* = **new** SingletonThesaurus();

*setSynomymPairs*(**new** HashMap<String, Set<String > > ( ));

}

**return** *thesaurus*;

The second used design pattern is the MVC. About it:

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.

In this design pattern the model role is of the SingletonThesaurus class.

**public** **class** SingletonThesaurus **implements** ThesaurusInterface {

**private** **static** **final** String ***XML\_FILENAME*** = "Thesaurus.xml";

**private** **static** SingletonThesaurus *thesaurus* = **null**;

**private** **static** Map<String, Set<String>> *synonymPairs*;

**private** SingletonThesaurus() {

}

**public** **static** **synchronized** SingletonThesaurus getInstance() {

**if** (*thesaurus* == **null**) {

*thesaurus* = **new** SingletonThesaurus();

*setSynomymPairs*(**new** HashMap<String, Set<String>>());

}

**return** *thesaurus*;

}

The View is actually the graphical user interface.

Snippet from it:

**public** **class** Gui {

**private** JFrame frame;

**private** JPanel panel;

**private** JButton addButton;

**private** JButton deleteWordButton;

**private** JButton deleteSynonymButton;

**private** JButton searchButton;

**private** JButton listButton;

**private** JTable table;

**private** JScrollPane scrollPane;

**public** Gui() {

frame = **new** JFrame("hg");

panel = **new** JPanel();

// frame.setLayout(null);

panel.setLayout(**null**);

addButton = **new** JButton("add");

deleteWordButton = **new** JButton("delete word");

deleteSynonymButton = **new** JButton("delete synonym");

searchButton = **new** JButton("search");

listButton = **new** JButton("list");

addButton.setBounds(10, 200, 200, 33);

deleteWordButton.setBounds(10, 250, 200, 33);

deleteSynonymButton.setBounds(250, 250, 200, 33);

searchButton.setBounds(250, 200, 200, 33);

listButton.setBounds(10, 296, 200, 33);

panel.add(addButton);

panel.add(deleteWordButton);

panel.add(deleteSynonymButton);

panel.add(searchButton);

panel.add(listButton);

table = **new** JTable();

table.setPreferredScrollableViewportSize(**new** Dimension(100, 50));

table.setFillsViewportHeight(**true**);

scrollPane = **new** JScrollPane(table);

panel.add(scrollPane);

scrollPane.setBounds(129, 11, 620, 149);

scrollPane.setViewportView(table);

frame.add(panel);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frame.setLocationRelativeTo(**null**);

frame.pack();

frame.setResizable(**true**);

frame.setBounds(400, 400, 780, 400);

frame.setVisible(**true**);

}

And the Controller implements the ActionListeners of the gui. It have one SingletonThesaurus object and also a gui one.

**public** **class** Controller {

**private** Gui gui;

**private** SingletonThesaurus thesaurus;

**private** Map<String, Set<String>> th;

**private** String[] coloumns;

**private** Object[][] data;

**public** Controller(Gui gui, SingletonThesaurus thesaurus) {

**this**.gui = gui;

**this**.thesaurus = thesaurus;

**this**.gui.addActionListenerToButtons(**new** ActionListenerToButtons());

}

It was also used the design my contract, pre conditions and post conditions and the invariant IsWellFormed() method;

/\*\*

\* Adds a new word

\*

\* **@param** word

\* **@param** synonym

\* **@pre** word != null

\* **@pre** synonym!= null

\* **@post** numberOfWords@pre +1 = numberOfWords

\* **@invariant** isWellFormed

\*/

**public** **void** addSynonym(String word, String synonym) {

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

**assert** word != **null** : "null word cannot be added";

**assert** synonym != **null** : "null synonym cannot be added";

**int** preNr = nrOfSynonyms(word);

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

Set<String> syns = *synonymPairs*.get(word);

syns.add(synonym);

**assert** nrOfSynonyms(word) == (preNr + 1) : "unsuccessful add";

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

updatingXML();

loadUpdated();

**return**;

}

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

newSyns.add(synonym);

*synonymPairs*.put(word, newSyns);

**assert** nrOfSynonyms(word) == (preNr + 1) : "unsuccessful add";

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

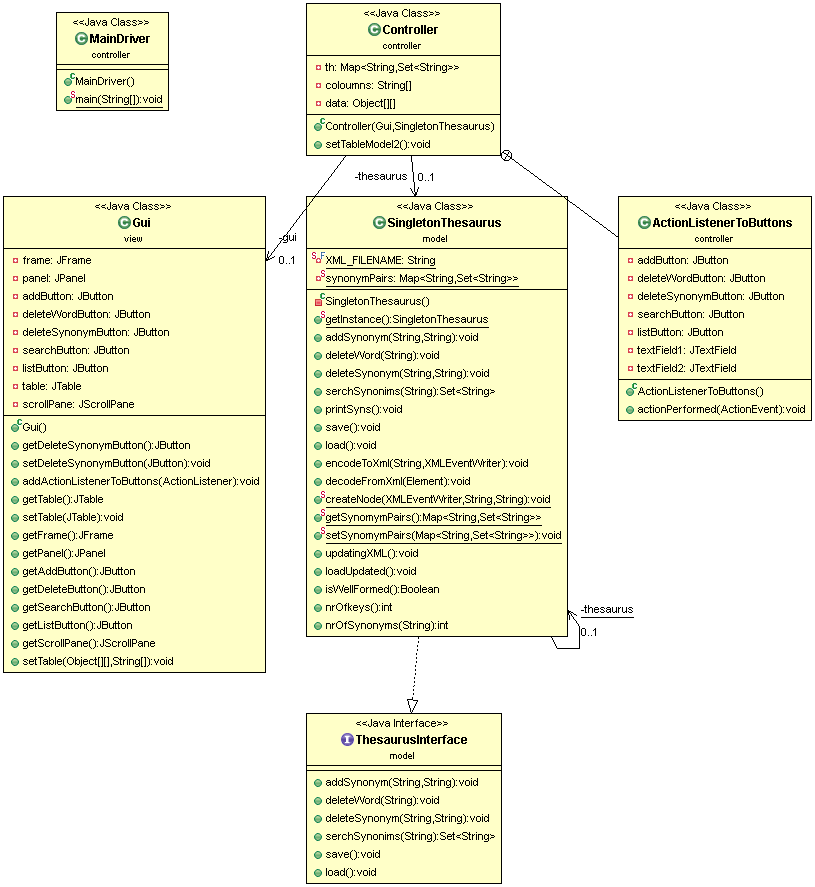
updatingXML();

loadUpdated();

}

1. Class Diagram

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .

. . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .

1. Future progress

It would be a great improvement to extend the functionality of the program. For example, to be able to search in an extended way, with more options and regex.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. .. . . . . . . . . . . . . . . . . . .. . .