Programming techniques

Homework 5: Simulating an application for a dictionary

Student: Sporis Ioan Catalin

Group: 30421

Contents

1. Project specification

The task is to create a java application implementing a 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.

2. Analysis of the problem, modelng application, scenarios, using cases

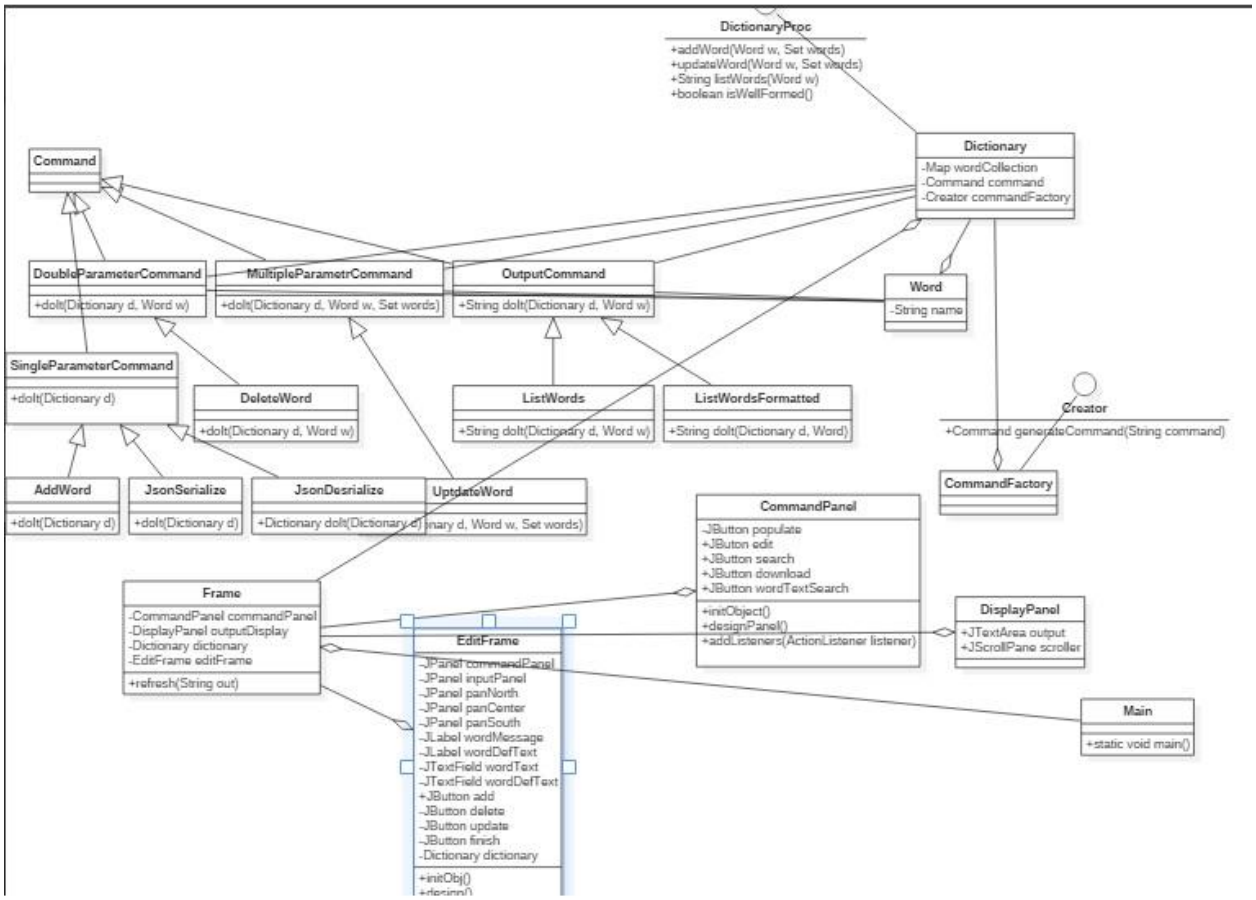
2.1. Analysis and modeling the problem

In order to obtain what we want we need a graphical user interface between the application and the user, such that this can communicate his demands. Also, we will need a series of algorithms to complete the demands.

The application will contain a frame called "**Frame**". In the frame there is a text field in order to insert the word that is searched for. There is a text area in order to display all the words existing in the list. There is also a button which make editing the definitions of one word possible, and also a button to save the changes in a file. Also in order to make the deletion possible a remove button must exist and also an add button for adding words.

3. Design

UML diagram



In the sense of design patterns, the ones I've used for my program would be :
Factory pattern and Command pattern;
In software engineering, a software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the

programmer can use to solve common problems when designing an application or system.

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply mutable state may be unsuited for functional programming languages, some patterns can be rendered unnecessary in languages that have built-in support for solving the problem they are trying to solve, and object-oriented patterns are not necessarily suitable for non-object-oriented languages.

Design patterns may be viewed as a structured approach to computer programming intermediate between the levels of a programming paradigm and a concrete algorithm.

Command decouples the object that invokes the operation from the one that knows how to perform it. To achieve this separation, the designer creates an abstract base class that maps a receiver (an object) with an action (a pointer to a member function). The base class contains an execute() method that simply calls the action on the receiver.

All clients of Command objects treat each object as a "black box" by simply invoking the object's virtual execute() method whenever the client requires the object's "service".

A Command class holds some subset of the following: an object, a method to be applied to the object, and the arguments to be passed when the method is applied. The Command's "execute" method then causes the pieces to come together.

Sequences of Command objects can be assembled into composite (or macro) commands.

On the other hand, the Factory Design Pattern is probably the most used design pattern in modern programming languages like Java and C#. It comes in different variants and implementations. If you are searching for it, most likely, you'll find references about the GoF patterns: Factory Method and Abstract Factory. Probably the factory pattern is one of the most used patterns.

Factory Method is to creating objects as Template Method is to implementing an algorithm. A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to subclasses that are supplied by the client. Factory Method makes a design more

customizable and only a little more complicated. Other design patterns require new classes, whereas Factory Method only requires a new operation.

People often use Factory Method as the standard way to create objects; but it isn't necessary if: the class that's instantiated never changes, or instantiation takes place in an operation that subclasses can easily override (such as an initialization operation).
Factory Method is similar to Abstract Factory but without the emphasis on families.
Factory Methods are routinely specified by an architectural framework, and then implemented by the user of the framework.

In my design, the Command Pattern is composed by the big abstract class Command, and the sub-classes that derive from it. In this manner, there are 4 classes first deriving from Command (DoubleParameterCommand, SingleParameterCommand, MultipleParameterCommand, OutputCommand), which allow the implementation of some concrete operations on a dictionary : AddWord, DeleteWord, JsonDeserialize, JSonSerialize, ListWords, ListWordsFormatted, UpdateWord.

In what Factory Pattern is concerned, I have created an interface Creator, with a single method designed to “create” an operation. Also, for this there was a need for creating a class too, so that the method could be implemented;

4. Results

The results have a designated area in the graphical user interface. By testing the right functioning of the application on various types of data, we can see that these are well performed. So the simulator has a good performance. Also the result can be immediately observed for the most of the operations. For example in case a new word is added or an existing word is deleted the results are available quickly. This fact is happening too when visualizing the words from the lists and also filtering the words after the search.

5. Conclusions

To sum up, the application implements successfully the task. Because of this project I learned to build more efficiently classes and instances of classes. Also I learned how to use the text to speech in java. The project can be improved by making the following:

- ✓ create a voice detector in order to record voices
- ✓ compare the pronunciation of the user with the pronunciation given by the application
- ✓ to create and select different types of dictionary like explicative dictionary or thesaurus