**Technical University of Cluj-Napoca  
Faculty of Automation and Computer Science  
Computer Science Department**

***Drimbarean Maria Cristina***

***Group:30425***



**Table of contents**

1. **Assignment objectives**

1.1General view

1.2 Personal approach

**2. The analysis**

2.1. Otput computations

2.2 Modeling

2.3. Scenarios

**3. Design**.

3.1. UML diagram

3.2. Data structures

3.3. Class projection

**4. Implementation and testing**

**5.Results**

**7. Conclusions**

**8. Bibliography**

**1.Assignment objective**

**1.1 General view**

*Task was described with the following statement:*

**Task description**

Consider a Dictionary of Synonyms 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\**.

*1.2Personal approach*

*I want to implement a application that will allow me to search and find words also with description, and also with a synonym list.*

**Application description**

One major advantage of having a dictionary of synonyms on the computer is the possibility of searching and finding quickly the word which is searched for.

***2.The analysis***

* 1. **General overview**
  2. **Input and Output**

The graphical user interface will provide the user not only with the possibility to read/ visualize data from the dictionary, but also with the option of introducing new words into the collection. One way of displaying words and synonyms onto the graphical user interface would be in form of a table. In this way, a user will have the possibility to display all the content of the collection. Moreover, when inputting new data to the dictionary, user will be prompted with a separate frame. Deleting an existing word is also a feature, so we will use a separate frame for choosing the word to be deleted.

***2.1Output computations***

**2.2Modeling**

**2.3Scenarios**

**2.3.1 Add word**

**Title**: Add word

**Resume**: The users inserts a new word into the application by filling in data in the three text fields from the graphical user interface corresponding to the word and to its synonym and also description.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can insert data
2. Normal scenario:

B1. User succeeds in inserting data into the desired text field;

B2. User selects the button for saving the properties;

B3. Data is validated and a new client object is added to the dictionary;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2.Application will see that the data is incorect and will wait for good input

**2.3.2 Delete word**

**Title**: Delete word

**Resume**: The users deletes a word from the dictionary application by filling in data: the word that is desired to be deleted.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in inserting data into the desired text field;

B2. User selects the button for saving the properties;

B3. Data is validated and a word is deleted from to dictionary;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2.Application will see that the data is incorect and will wait for good input

**2.3.3 Search word**

**Title**: Search word

**Resume**: The users inserts a word into the application by filling in data.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in inserting data into the desired text field;

B2. User selects the button for searching the word;

B3. Data (synonym ) is validated and displayed onto the screen;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2.Application will see that the data is incorect and will wait for good input

**2.3.4. Verify Dictionary Consistency**

**Title**: Verify Dictionary Consistency

**Resume**: The application verifys the consistency of the dictionary and displays in the graphical user interface.

**Actors**: User, system.

**Scenario**:

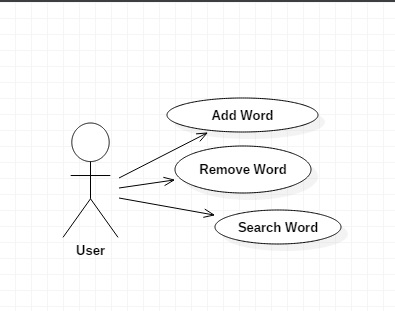
1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1.State of consitency is diplayed in the bottom label.

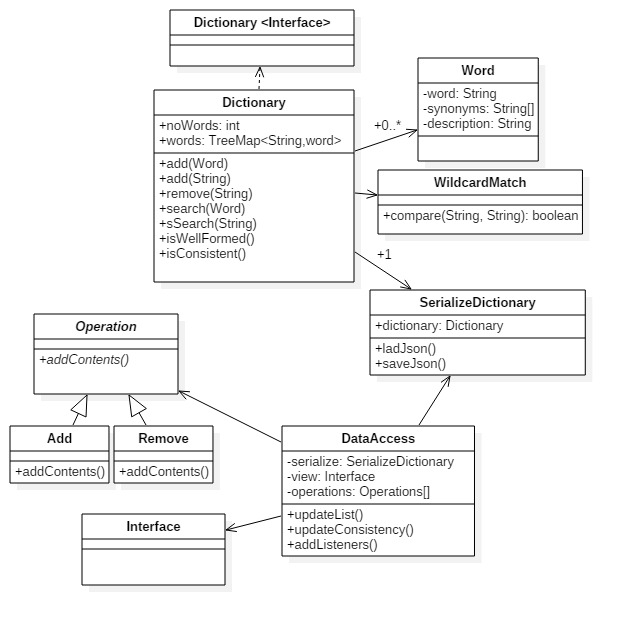
**3.Projection**

3.1 UML Diagrams

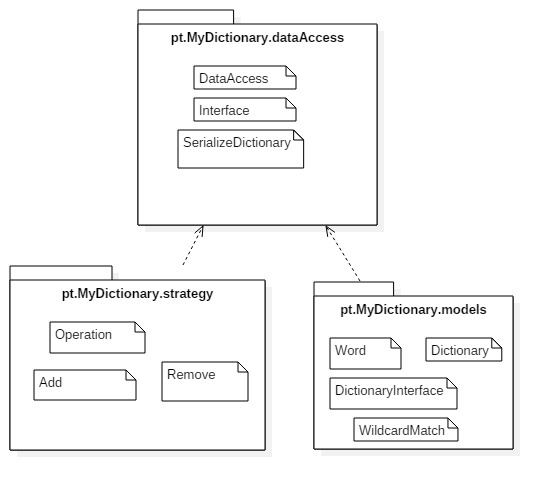
a)Use case



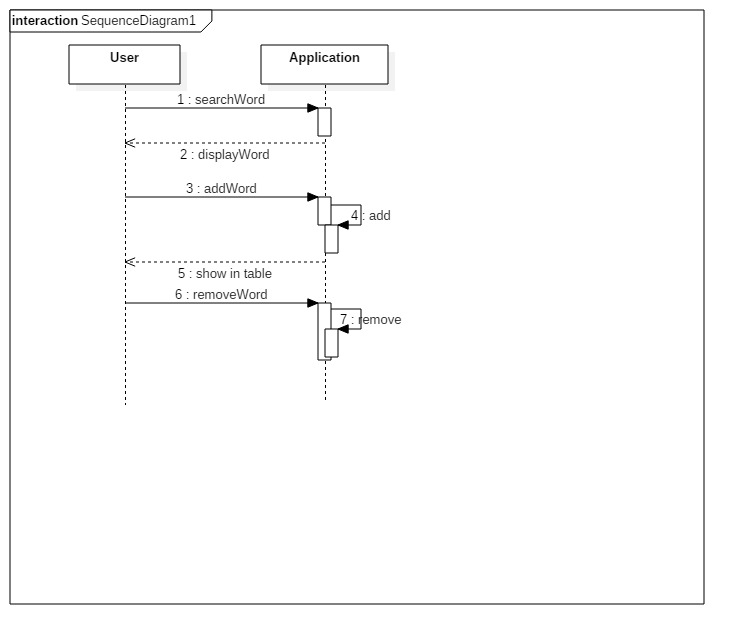
b)Class diagram



c)Package Diagram



d)Sequence diagram



**3.2Data structures**

The main problem to be taken into account when designing a class is choosing which data structures are needed.

1. **Arrays**

This Java class (member of the Java Collection Framework) has many methods for manipulating arrays (such as sorting and searching).

1. **Array List**

The Java class (member of the Java Collection Framework) has resizable implementation of the List interface, implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

*Advantages:* Array List can be used to store the list of clients. Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.

Methods like add or remove (remove an element from a specific position in the array list), size (returns the number of elements), get (returns the element at the specific index from the array) and constructors for the array list from class ArrayList can be used.

*Disadvantages:* Array List is not synchronized, but it can be synchronized using the Java synchronize.

1. **Hash Table**

The Java Class Hash Table implements a hashtable, which maps keys to values.In this class, any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method. An instance of Hashtable has two parameters that affect its performance: *initial capacity* and *load factor*. The *capacity* is the number of *buckets* in the hash table, and the *initial capacity* is simply the capacity at the time the hash table is created. The hash table is *open*: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The *load factor* is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling there hash method.

1. **Tree Map**

A Red-Black tree based [NavigableMap](https://docs.oracle.com/javase/7/docs/api/java/util/NavigableMap.html) implementation. The map is sorted according to the [natural ordering](https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html) of its keys, or by a [Comparator](https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the containsKey, get, put and remove operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's Introduction to Algorithms.

Note that the ordering maintained by a tree map, like any sorted map, and whether or not an explicit comparator is provided, must be consistent with*equals* if this sorted map is to correctly implement the Map interface. (See Comparable or Comparator for a precise definition of consistent with equals.) This is so because the Map interface is defined in terms of the equals operation, but a sorted map performs all key comparisons using its compareTo (or compare) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map is well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the Map interface.

Advantage: Provides constant time access for get, put, and remove operations.

Disadvantage: **Note that this implementation is not synchronized.** If multiple threads access a map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with an existing key is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the[Collections.synchronizedSortedMap](https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedSortedMap(java.util.SortedMap)) method.

**Chosen data structure:** TreeMap

TreeMap implements the Map interface, together with Serializable and Clonable, so it respects the specific requirements. In this application , it is good for us to be able to map a key to a value, you need a null key also.

**3.3 Class projection**

In this section i will present the design Paterns used:

**Composite**

**Intent**

Compose objects into tree structures to represent part-whole hierarchies.

Composite lets clients treat individual objects and compositions of objects

uniformly.

**Motivation**

Graphics applications like drawing editors and schematic capture systems let users

build complex diagrams out of simple components. The user can group components

to form larger components, which in turn can be grouped to form still larger

components. A simple implementation could define classes for graphical primitives

such as Text and Lines plus other classes that act as containers for these

primitives.

But there's a problem with this approach: Code that uses these classes must treat

primitive and container objects differently, even if most of the time the user

treats them identically. Having to distinguish these objects makes the application

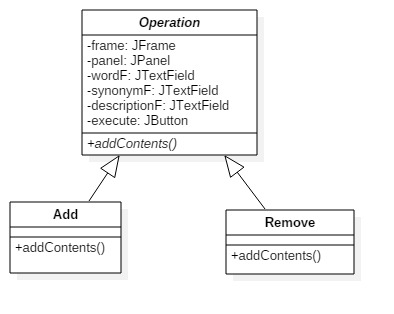
more complex.

Here is a class diagram that shows an implementation of this design pattern:

In my design i also used this approach as i needed a neu window/frame that will alllow me to introduce the input and by knowing wich implementation i have provided make the corespondent operation in the Dictionary class.



My class diagram for this pattern is almost similar and it is made of three classes. One of them is the abstract class Operation and the other to are concrete implementations. It saves the developer a lot of code to write as the to operations are almost the same, and you have to implement only the parts that you use, and the action listener for the button.



**Data Access Object Pattern**

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

* **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
* **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
* **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

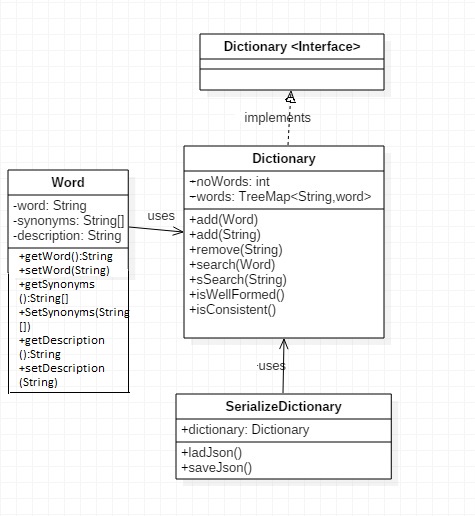
## **Implementation**

We are going to create a *Student* object acting as a Model or Value Object . *StudentDao* is Data Access Object Interface . *StudentDaoImpl* is concrete class implementing Data Access Object Interface .  *DaoPatternDemo*, our demo class, will use *StudentDao* to demonstrate the use of Data Access Object pattern.

The following diagram represents an example of implementation , you can observe the interface that uses the Data object named student Who just has some iportant data, and the concrete implementation that can manages a list of Data objects and performs the simple operations like add and delete from storage collection.



In my own implementation I used the same approach, as the most valuable object is the Word, wich has all the info i need and the rest of the classes are used just to provide user access to information. The following diagram shows my own implementation with the dictionary interface and also the concrete class Dictionary, witch also uses Design by Contract.



The Word class contains simple getters and setters for the data needed, The interface enables design by contract and external manipulations, and the dictionary class has the methods needed to manipulate the storage.

**4.Implementation an testing**

I will show here the format of the graphic user interface.

**5.Results**

A database of words is stored and you are able to find the descriptions and the synonyms of the words searched. Also as for the implementation this simulation provided a good overview on how to design and how to use some design patterns to solve your problems. I learned some specific things: Designing object-oriented software is hard, and designing *reusable*

object-oriented software is even harder. You must find pertinent objects, factor

them into classes at the right granularity, define class interfaces and inheritance

hierarchies, and establish key relationships among them. Your design should be

specific to the problem at hand but also general enough to address future problems

and requirements. You also want to avoid redesign, or at least minimize it.

Experienced object-oriented designers will tell you that a reusable and flexible

design is difficult if not impossible to get "right" the first time. Before a

design is finished, they usually try to reuse it several times, modifying it each

time.

**6.Conclusions**

**Further development**:

By means of using serialization and a friendly graphical user interface, a user-friendly application which simulates the functionality of a dictionary has been developed. However, there are further improvements that can be done, in order to adapt the application to real-world situations. Moreover, this small application can be modified into an applet in order to be published onto a website.

7. *Bibliography*

jSON- http://www.json.org/

https://sites.google.com/site/gson/gson-user-guide

http://wiki.fasterxml.com/JacksonInFiveMinutes

Design Patterns

<http://www.tutorialspoint.com/design_pattern/>

**Design Patterns:Elements of Reusable Object-Oriented Software**