Mincu Mihai-Daniel

Google Translate Who Concept Project

As the title might suggest, this project has the goal to put a base of an API who can become even more effective than Google Translate. In the same time, the API will manage dictionaries in **JSON** format, dictionaries that will be utilized in the translating process as well.

The improvement from Google Translate I addressed is providing more than just one translation for a specific sentence by replacing specific words with their synonyms.

I have chosen to implement this API in Java because I strongly believe this programming language provides one of the most organized code structure that always helps the programmer as well as potential customers have a proper and brief understanding of the concept.

This is the format a word appears in the JSON Romanian Dictionary:

Line number	RO Dictionary Example	Field description
1	{	A JSON object is placed between {} and a JSON list is placed between [].
2	”word”: ”fibră”,	The word in romanian.
3	“word_en”: “fiber”,	The word translated in english, available in every translation dictionary.

4	”type”: ”noun”,	Part of speech – for now I only use nouns and verbs.
5	”singular”: [”fibră”],	Singular form
6	”plural”: [”fibre”],	Plural form
7	”definitions”: [	List of definitions for the current word
8	{	
9	“dict”: “Dicţionar de sinonime”,	Name of the dictionary the definition comes from
10	“dictType”: ”synonyms”,	Type of the dictionary – can be synonyms or definitions
11	”year”: ”1998”,	Year the dictionary was published
12	”text”: [”filament”, ”fir”]	Actual definition or synonym list, in this case we have a list of 2 synonyms
13	},	
14	{	
15	“dict”: “DEX ‘09”,	Name of the dictionary the definition comes from
16	“dictType”: ”definitions”,	Type of the dictionary
17	”year”: ”2009”,	Year the dictionary was published
18	”text”:[”Fir subțire, netors, de proveniență vegetală, animală sau minerală ori produs pe cale sintetică, folosit de obicei ca materie primă la fabricarea țesăturilor.”, ”Celulă vegetală alungită situată în țesutul lemnos.”]	Actual definition or synonym list, in this case we have a list of 2 definitions
19	}	
20	]	
21	}	

Note: Depending on the part of speech, a noun has its only singular form as singular and its only plural form as plural, while a verb has its first, second and third person singular forms as singular and its first, second and third person plural forms as plural. This explains why the „singular" and „plural" fields are lists.

In order to work with JSON formats, I decided to use the **gson** library. For example, the library and be imported in IntelliJ IDEA software by adding from Open Module Settings by right click on the current directory -> Library -> From Maven, com.google.code.gson:gson:2.8.5 or any version you choose to use.

My main entities (classes) I work with are Word and Definition. Both have the attributes according to the fields in the JSON format describing them. The Word class impements the Comparable interface in order to facilitate the words being sorted alphabetically with ease. The Definition class also implements the interface because I have chosen to sort them by the year of the dictionary.

Word class :

```
public class Word implements Comparable<Word> {
    String word;
    String word_en;
    String type;
    ArrayList<String> singular;
    ArrayList<String> plural;
    ArrayList<Definition> definitions;
```

Definition class:

```
public class Definition implements Comparable<Definition> {
    String dict;
    String dictType;
    int year;
    ArrayList<String> text;
```

I implemented 10 methods in a Methods class:

1) public static void getDataCollection()

I decided to keep the data in a HashMap, the key being the word with all its information and the value being the language of the word. The method selects only the .json files from the Dictionaries directory and parses the JSON dictionary into an ArrayList of Words, then the hasmap entires are created. The parsing was done by using the type ArrayList<Word> as typetoken in a fromGson() function call.

2) public static boolean addWord(Word word, String language)

Adds a word in the map if not already existent. It returns True or False based on its success.

3) public static boolean removeWord(String word, String language)

Removes the given word from the map, only if it exists. It returns True or False based on its success.

4) public static boolean addDefinitionForWord(String word, String language, Definition definition)

If the given word exists, the specified definition is added but only if the word has no other definition from the same dictionary. It returns True or False based on its success.

5) public static boolean removeDefinition(String word, String language, String dictionary)

If the given word exists, we search for the definition and if the definition exists, we remove it. It returns True or False based on its success.

6) public static String translateWord(String word, String fromLanguage, String toLanguage)

A word translates another word if and only if they have the same english translation. So, if the word to be translated exists in the fromLanguage, we search in the hash map for the word that has the same english translation an is in the toLanguage. If the word to be translated does not exist, the method returns null. If the word does not have translation, it will not be translated. Otherwise it will be translated and kept in the same form, so this is why we need to find the correct form as well (first person plural is translated to first person plural).

7) public static String translateSentence(String sentence, String fromLanguage, String toLanguage)

By tokenizing, I obtain the words to be translated by calling the previous function. The method returns null if not every word if the

sentence has a translation, otherwise it returns the translated sentence.

8) public static ArrayList<String> translateSentences(String sentence, String fromLanguage, String toLanguage)

This method will return the default translation option for the sentence (by calling method 7) and it will also look for synonyms for every word of the sentence, so at most two other translation options will be provided.

9) public static ArrayList<Definition> getDefinitionsForWord(String word, String language)

The method gets the definitions for the given word sorted by the year attribute of the Definition class. I use a Priority Queue, then the elements of the Priority Queue go in an Arraylist of Definitions.

10) public[1] static void exportDictionary(String language) throws IOException

The eventually modified dictionary for the given language is exported in the JSON format, I used the toJson() function provided by the gson library, along with the other neccesary characters. The format of the dictionaries may also be improved in the future. The dictionary is saved in a .json file.

I have chosen to test every method in a Test class, with both base cases and exception cases. I use a Romanian dictionary as well as a French one, a German one and a dictionary for an invented language that codes a romanian word with the exact keys pressed on a classic phone keyboard. I add 0-s for special characters. There is a null dictionary test as well.

[1] Word formatting☺