

CSE-3216: Software Design Patterns

Assignment 02

Submitted to:

Dr. Mohammad Samiullah

Assistant Professor

Department of Computer Science and Engineering

University of Dhaka

Submitted by:

Kazi Shadman Sakib

Roll: FH-97

3rd Year

Department of Computer Science and Engineering

University of Dhaka

**Scenario:**

Sentence Generator is a program that generates sentences using English words. All sentence generators have an internal vocabulary (one vocabulary for each sentence generator), which is initially empty and can be updated by adding some words. All sentence generators are able to generate sentences according to some rules. You have to implements following three types of sentence generators:

**Random Sentence Generator - RSG**

When a word is added to RSG, before storing it in internal vocabulary, the word is

converted to lowercase. RSG generates new sentences by randomly picking random

amount of words from its internal vocabulary and concatenating them using single space between the words.

**Sorted Sentence Generator - SSG**

Before adding a word into its internal vocabulary, SSG also converts the word to

lowercase. Like RSG, SSG picks up the words randomly. The only difference is that it

sorts these words before the concatenation.

**Ordered Sentence Generator - OSG**

OSG is different from both RSG and SSG. A word, before adding into internal

vocabulary, will be converted to upper case and reversed. OSG concatenates all of the

words in the same order they have been added to the vocabulary.

Finally, after creating these sentence generators we have to create a sentence generator application(console/gui/etc), in which the user can choose a particular sentence generator from a menu to create a sentence. Moreover, using the same menu user can also give input to add new words in the vocabulary.

**Classes build:**

* ISentenceGenerator : Interface
* RandomSentenceGenerator : Class
* SortedSentenceGenerator : Class
* OrderedSentenceGenerator : Class
* SentenceGeneratorAdapter : Class
* SentenceGenerator97 : Class

**Patterns Used and Assumptions Made:**

Strategy pattern was used while designing this assignment. It is a behavioral software design pattern that enables selecting an algorithm at runtime from a family of algorithms.

We have first built an interface named ISentenceGenerator, which has two public abstract methods named addWordToVocabulary() and generateSentence().

This ISentenceGenerator interface is then implemented by three classes named RandomSentenceGenerator, SortedSentenceGenerator and OrderedSentenceGenerator.

These three classes override the methods of the interface ISentenceGenerator.

**Assumptions in RandomSentenceGenerator:**

RandomSentenceGenerator is a singleton class too and it also implements the interface named ISentenceGenerator, which has one static object of its own named “instance”, we have built it like this because we will use this same object continuously while being in the runtime of the application. This “instance” object of RandomSentenceGenerator will check if the object of RandomSentenceGenerator was created already or not, if it was already created then it will return the “instance”, if it was not already created then it will create a new RandomSentenceGenerator() and return the “instance”.

In the addWordToVocabulary(String words) method of RandomSentenceGenerator, we have first converted the whole input words to lowercase. Then we splitted the words to a string array named wordsToBeAddedRSG. Thus, with a for loop we have added the wordsToBeAddedRSG (words) to the internal vocabulary. Before adding the words to the internal vocabulary we have checked if the internal vocabulary already has the word or not. If it has the word already we did not add the word and continued. But if it does not have the word we added it to the internal vocabulary.

In the generateSentence() method of RandomSentenceGenerator, we have used the random function for getting a random number for the sentence generator that generates our sentences where minimum number of word would be 1 and maximum number of word would be internalVocabularyRSG.size() + 1. We have also checked if the size of internalVocabularyRSG is greater than 0 or not. If it's greater than 0 then we approach the random function for getting a random number. If it is not greater than 0 then we simply assigned randomAmountOfWordsRSG to 0 and returned the empty generatedSentenceRSG.

Thus, by using a for loop we concatenate the generated words to a sentence and return the value of generatedSentenceRSG. Here, we have assumed that a generated sentence can have only one word too as the random function’s range is from 1 to

internalVocabularyRSG.size() + 1.

**Assumptions in SortedSentenceGenerator:**

SortedSentenceGenerator is a singleton class too and it also implements the interface named ISentenceGenerator, which has one static object of its own named “instance”, we have built it like this because we will use this same object continuously while being in the runtime of the application. This “instance” object of SortedSentenceGenerator will check if the object of SortedSentenceGenerator was created already or not, if it was already created then it will return the “instance”, if it was not already created then it will create a new SortedSentenceGenerator() and return the “instance”.

In the addWordToVocabulary(String words) method of SortedSentenceGenerator, we have first converted the whole input words to lowercase. Then we splitted the words to a string array named wordsToBeAddedSSG. Thus, with a for loop we have added the wordsToBeAddedSSG (words) to the internal vocabulary. Before adding the words to the internal vocabulary we have checked if the internal vocabulary already has the word or not. If it has the word already we did not add the word and continued. But if it does not have the word we added it to the internal vocabulary.

In the generateSentence() method of SortedSentenceGenerator, we have used the random function for getting a random number for the sentence generator that generates our sentences where minimum number of word would be 1 and maximum number of word would be internalVocabularySSG.size() + 1. We have also checked if the size of internalVocabularySSG is greater than 0 or not. If it's greater than 0 then we approach the random function for getting a random number. If it is not greater than 0 then we simply assigned randomAmountOfWordsSSG to 0 and returned the empty generatedSentenceSSG.

Thus, by using a for loop we concatenate the generated words to a sentence and return the value of generatedSentenceSSG. Now sorting will be done according to the sentence generated randomly which is assigned to generatedSentenceSSG. We split the words of generatedSentenceSSG to a String array named generatedSentenceWord. Then, we add these String arrays to an Array list named sortGeneratedSentenceSSG. Then we used the Collections.sort() which sorts the words from the array list. Thus we again use a for loop to concatenate the sorted words to generatedSentenceSSG.

Here, we have assumed that a generated sentence can have only one word too as the random function’s range is from 1 to internalVocabularyRSG.size() + 1.

**Assumptions in OrderedSentenceGenerator:**

OrderedSentenceGenerator is a singleton class too and it also implements the interface named ISentenceGenerator, which has one static object of its own named “instance”, we have built it like this because we will use this same object continuously while being in the runtime of the application. This “instance” object of OrderedSentenceGenerator will check if the object of OrderedSentenceGenerator was created already or not, if it was already created then it will return the “instance”, if it was not already created then it will create a new OrderedSentenceGenerator() and return the “instance”.

In the addWordToVocabulary(String words) method of RandomSentenceGenerator, we have first converted the whole input words to uppercase. Then we splitted the words to a string array named wordsToBeAddedOSG. Thus, with a for loop we have added the wordsToBeAddedOSG(words) to the internal vocabulary. Before adding the words to the internal vocabulary we have reversed every word from the wordsToBeAddedOSG array by using StringBuilder. We have also checked if the internal vocabulary already has the word or not. If it has the word already we did not add the word and continued. But if it does not have the word we added it to the internal vocabulary.

In the generateSentence() method of OrderedSentenceGenerator, we have simply concatenated the words using a for loop to generatedSentenceOSG.

Thus, by using a for loop we concatenate the generated words to a sentence and return the value of generatedSentenceRSG. Here, we have assumed that a generated sentence can have only one word too as the random function’s range is from 1 to

internalVocabularyRSG.size() + 1.

**Assumptions in SentenceGeneratorAdapter:**

SentanceGeneratorAdapter is a helper class which helps to process all the algorithms of the Sentence Generator. In this class the users can choose which instructions should be done (Add words, Generate sentence or Exit). This class takes input from the user for adding words. This class also ensures if the internal vocabulary of any Sentence Generator is empty or not. If it is empty then it tells the users to add words. If not then it generates the sentences using the chosen algorithm for sentence generating.

**Assumptions in SentenceGenerator97:**

SentenceGenerator97 class selects the required algorithm from a family of algorithms (RandomSentenceGenerator, SortedSentenceGenerator, OrderedSentenceGenerator). The selection is done by the users from the User Interface. This class uses SentanceGeneratorAdapter to process the required algorithm further.