CIT594 Group Project Milestone 1

Group Members:

Jiyun Duan, Kengpian Huang, Zhou Luo, Ke Zhu

Summary

Vocabulary building has always been a barrier that international students have to overcome if they want to excel in the English language proficiency tests such as "Test of English as a Foreign Language"(TOEFL) as per the admission requirements. Also, even if for English native speakers, memorizing GRE vocabularies requires hard work. Therefore, to meet the demand, our group decided to build an app that facilitates the memorizing process of English words.

Description

I. Data

Based on the users' preferences, our app would choose corresponding word banks such as GRE or TOEFL.

II. Features

(1) Flashcards

In the format of flashcards, words would be shown to users. Users are allowed to mark the word as "known" or "unknown". "Known" words will be deleted from that user's word list. For "unknown" words, their definition would automatically show up, and these words would re-appear for users to review after a certain amount of time, according to the forgetting curve.

(2) Promote High-frequency words

We will build individual max heaps for the different word banks based initially on the priority of each word. The priority is associated with three factors.

- The first factor is the occurrence frequency of the word, indicating its importance in each exam.
- The second factor is the users' familiarity with the word.

  For example, if we show a user the word "banana", and the user chooses to click "known". Then the word would be deleted from the heap and get stored in a "known" list.

If we show a user the word “philanthropy”, and the user chooses to click “unknown”, then the value of the would increase and get prioritized. And next time, the user comes across the word “philanthropy” again and chooses “known”, we would reduce the value of the word, still keep it in the max heap, but lower its priority. If the user keeps choosing “known” every time he/she sees the word, we would delete the word from the heap after a certain threshold.

- The third factor is time. The same word would not keep appearing in a short period of time even if it is “unknown” by the user. We create a *timer* attribute for the word node in the max heap. Only if the time threshold is reached, the word would be able to show up.

(3) Customization

Users are allowed to check the list of words that they have encountered but not mastered. Also, they are able to view the list of deleted words, which they have already mastered. The app would allocate an appropriate workload according to different users’ learning goals.

(4) Ranking

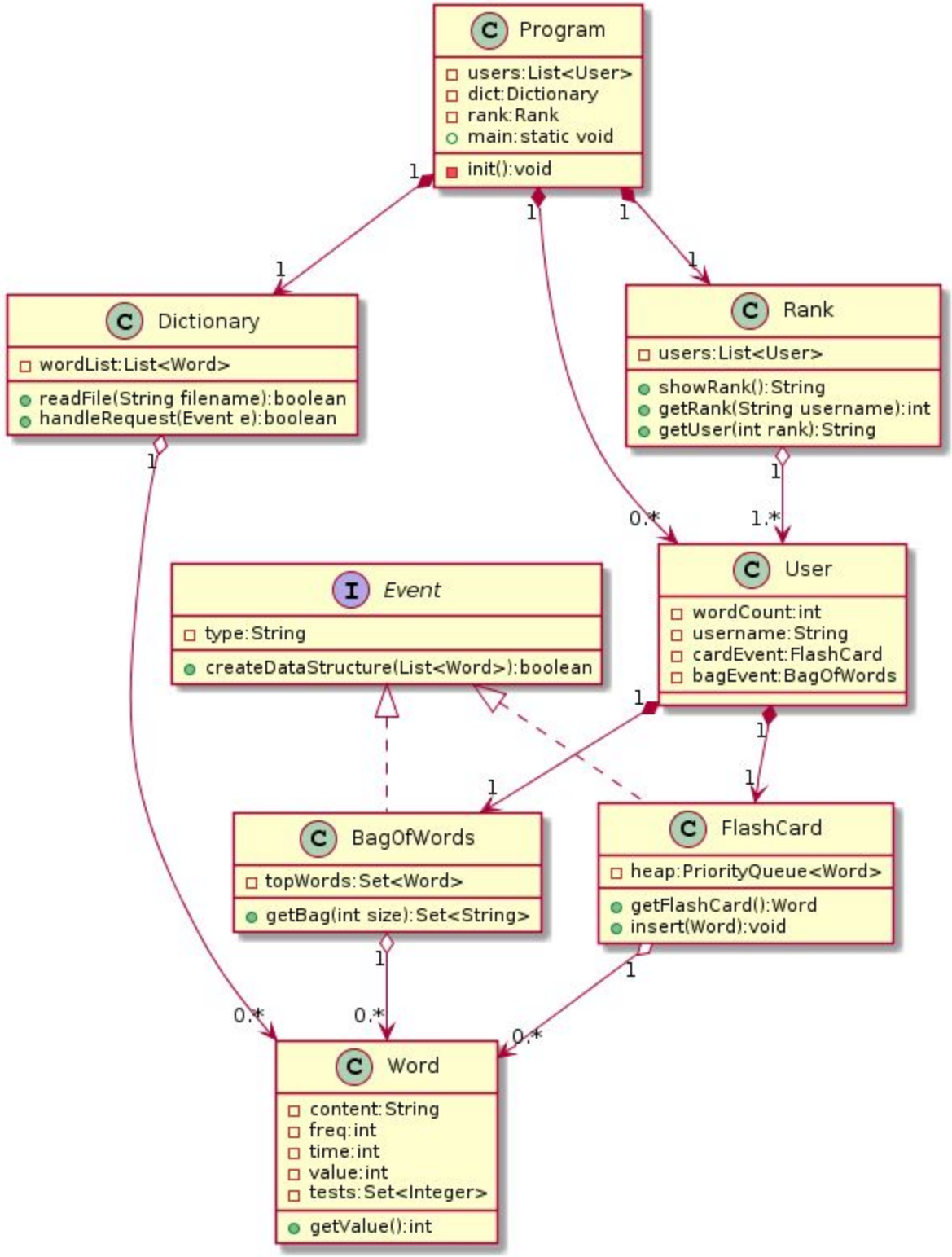
A ranking system would be built where users are ranked in the order of the number of words they master each day.

(5) Bag of Words

By counting the words’ frequency over all text from the same type of exams, we get a list of the most frequent words (the list size is a constant, ***LIST_SIZE***, it could be 30). We would retrieve a bag of frequent words (***BAG_SIZE*** is a constant determined by the user) from the list by counting the frequency of the whole combination over several tests.

(Instead of computing the frequency of all possible permutation of ***BAG_SIZE*** words, we choose to find out the most frequent permutation of words among a list of frequent words. This is due to the consideration that it takes a long time to compute the frequency of all possible permutations over all words in the vocabulary list. Besides, when unrelated words happen to show up together a lot, it neither means that this combination is a highly-frequent phrase nor each word from the combination is such a frequent word that we should prioritize it. So to make our project more practical, we are thinking about counting the frequency of every possible combination of ***BAG_SIZE*** words from the top ***LIST_SIZE*** frequent words and offer users the most frequent bag of words.)

Class Diagram



Design Pattern And Data structures Used

Design Patterns

I. **Visitor**

In our dictionary, we want to traverse it in different ways, and the basic traversal will provide an iterator of reference to each word.

(1) Heap Construction

If we want to add words to the heap, then we loop through all the words and then add each Word in the list to the heap. Also, we'd update the weight of Word according to timestamp, word's original frequency, and user's feedback. After updating the weight, the Word would be inserted back to the heap.

(2) Print Words

If we just want to see what words are in this list, then we loop through the list and print all the words.

(3) Most Frequent Words

If we want to retrieve top 5 / top 10 / top 20 most frequent words in a single passage, then we need to loop through the list and sort them according to their frequencies. Based on the users' need, we then return a certain amount of most frequent words.

(4) User Decides Behavior

We provide two functions: flashcard and a bag of words. With "Flashcard" function, Words are picked according to their weights and then sent to users one by one through flashcard. With "bag of words" function, we offer users a bag of frequent words, in which the permutation has the highest frequency.

Data structures

I. **Max-Heap**

We use Max-heap to store Words sorted by their weights. When users use the flashcard function, the Word with the largest weight would be popped and its value would be updated according to users' performance. If the user chooses "Known", the known words would be added into "Delete" set. Otherwise, the Word would be inserted back into the heap.

II. **Set**

We will use set to store Word that has been mastered by the user. Users are allowed to print the list and retrieve the most recently memorized word.

III. **LinkedList**

LinkedList would be used when reading file and processing text.