Cheater’s Hangman

Premise

Playing a game relies on trust. We intrinsically assume that if we are playing a game against someone, then the rules will be followed. In other words, people aren’t prepared when someone might be cheating at a game, which makes it an excellent strategy (if you prepared to accept the consequences).

Your goal is to write a computer program that allows a user to play a game of Hangman, but with a hidden twist. If you don’t know what Hangman is, go look it up on Wikipedia and play a few games on a whiteboard or a piece of paper with your friends.

A core assumption in hangman is that the person who knows the word does not change the hidden word. Our program is going to do that and do it without getting caught. Here’s an example of how this might work.

Suppose the player has made a lot of guesses and has only one wrong guess left. The player’s guesses have revealed:

```
GOA_
```

Based on the player’s previous guesses and knowledge of the English language, there are only two possible words left: “GOAD” and “GOAL.” In a normal game of hangman, where the person with the hidden word is playing according to the rules, this means the player has a 50% of guessing the right word.

In our game, the computer will cheat: if the player guesses the blank letter is “L,” the computer will say something to the effect of “Sorry, you’re out of guesses! You lose! The hidden word was “GOAD.”” If the player guesses “D” instead, the computer pretends that the hidden word “GOAL” the whole time. In other words, the player will always, always, always lose in this scenario because the computer will cheat about what the hidden word was.

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class hangman {
    public static void main(String[] args) throws FileNotFoundException {
        int wordLength;
        Set<String> guesses = new HashSet<>();
        Scanner readUser = new Scanner(System.in);
        //get word length
        System.out.println("Enter number of letters for word");
        wordLength = readUser.nextInt();
        List<String> list = getList(wordLength);
        //get valid input
        while (list.isEmpty()) {
            System.out.println("Please enter valid number of letters for 
word");
            wordLength = readUser.nextInt();
            list = getList(wordLength);
        }

        System.out.println("How many times would you like to guess?");
        int guessLength = readUser.nextInt();
        //get guesses
        for (int i = 0; i < guessLength; i++) {
            System.out.println("Enter your letter guess");
            String guess = readUser.next().toLowerCase();
            if(guesses.contains(guess)){
                System.out.println("You already guess that letter. Please 
enter a new letter");
               guess = readUser.next().toLowerCase();
            }
            guesses.add(guess);
            HashMap<String, List<String>> families = wordFamilies(list, 
guesses);

            String largest = largestWordFamily(families);
            System.out.println(largest);
            list = families.get(largest);

            if(list.size() == 1){
                System.out.println("You won!");
                break;
            }
        }
        System.out.println(list.get(0));
        System.out.println("Would you like to play again?");

    }
    public static List<String> getList(int wordLength) throws 
FileNotFoundException {
        List<String> list = new ArrayList<>();
        Scanner readTxt = new Scanner(new File("words.txt"));
        while (readTxt.hasNext()) {
```

```
      String word = readTxt.nextLine();
      if (word.length() == wordLength) {
        list.add(word.toLowerCase());
      }
    }
    return list;
  }
  public static HashMap<String, List<String>> wordFamilies(List<String>
list, Set<String> guesses){
    HashMap<String, List<String>> wordFamilies = new HashMap<>();
    for (int i = 0; i < list.size(); i++) {
      String key = makeKey(list.get(i), guesses);
      if(!wordFamilies.containsKey(key)){
        List<String> newlist = new ArrayList<>();
        newlist.add(list.get(i));
        wordFamilies.put(key, newlist);
      }else{
       wordFamilies.get(key).add(list.get(i));
      }
    }
    return wordFamilies;
  }
  public static String makeKey(String word, Set<String> guesses){
    String key = "";
    for (int i = 0; i < word.length(); i++) {
      if(guesses.contains(word.substring(i,i+1))){
        key += word.charAt(i);
      }else{
        key += "_";
      }
    }

    return key;
  }
 public static String largestWordFamily(HashMap<String, List<String>>
wordFamilies){
   int largest = 0;
   String largestKey = "";
   for (String key: wordFamilies.keySet()){
     if(wordFamilies.get(key).size() > largest){
       largestKey = key;
       largest = wordFamilies.get(key).size();
     }
   }
   return largestKey;
 }

}
```