Homework Turnin

Name:	Xuqing Wu
Account:	xw88 (xw88@uw.edu)
Student ID:	1933202
Section:	AD
Course:	CSE 143 20wi
Assignment:	a4
Receipt ID:	55de2f21a3dd46019ceb3bf5692fa89b

Turnin script completed with output:

Turnin Successful!

The following file(s) were received:

HangmanManager.java (5197 bytes, sha256: ed0ddf4b1f853db60b7a6cf356ee96c0)

```
// Xuqing Wu
// 2/5/2020
// CSE143
// TA: Eric Fan
// Assignment #4
//
// Class HangmanManager allows a client to manage a game
// called eveil hangman and keep track of the state of the game
// and cheat by delaying picking a word until it is forced to.

import java.util.*;

public class HangmanManager {
   private int chance;  //the chance of guess left
   private String wordFamily;  //the current right pattern that is guessed
   private Set<String> wordRemain;  //set of strings that can be picked
   private Set<Character> guessed;  //characters that has been guessed

   //pre: parameter length is bigger than 0 and max is bigger than or equal 0
   //     (throw IllegalArgumentException if not)
   //post: initiate all the fields. Put all words of given length from
   //      dictionary into set of words and eliminate duplicates
   //      Collection<String> dictionary: a file with all words
   //      length: the length of word that is guessed
   //      max: time that player can guess at most
   public HangmanManager(Collection<String> dictionary, int length, int max) {
      if(length < 1 || max < 0) {
         throw new IllegalArgumentException();
      }
      chance = max;
      wordFamily = "";
      for(int i = 0; i < length; i++) {
         wordFamily += "- ";
      }
      wordRemain = new TreeSet<>();
      for(String word: dictionary) {
         if(word.length() == length) {
            wordRemain.add(word);
         }
      }
      guessed = new TreeSet<>();
```

```java
    }

    //post: return the current set of words that computer can choose
    public Set<String> words() {
       return wordRemain;
    }

    //post: return the number of guesses the player has left
    public int guessesLeft() {
       return chance;
    }

    //post: return the current set of letters that have been guessed
    public Set<Character> guesses() {
       return guessed;
    }

    //pre: the set of words that can be chosen is not empty
    //     (throw IllegalStateException if not)
    //post: return the current pattern of right guesses
    //      Letters that have not been guessed are displayed as dashes
    //      and there are spaces to separate letters
    public String pattern() {
       if(wordRemain.isEmpty()) {
          throw new IllegalStateException();
       }
       return wordFamily;
    }

    //pre: number of guesses left is bigger than or equal to 1 and current
    //     set of words that computer can choose is not empty
    //     (throw IllegalStateException if not)
    //     character passed as parameter was not guessed previously
    //     (throw IllegalArgumentException if not)
    //post: record the next guess made by the user by deciding which
    //      set of words can be chose. Return the number of occurrences of
    //      the guessed letter in the new pattern and update the number
    //      of guesses left
    public int record(char guess) {
       if(chance < 1 || wordRemain.isEmpty()) {
          throw new IllegalStateException();
       }
       if(guessed.contains(guess)) {
          throw new IllegalArgumentException();
       }
       guessed.add(guess);
       Map<String, Set<String>> map = returnMap(wordRemain, guess);
       changeWordRemain(map);
       int occurrence = changeWordFamily(map, guess);
       if(occurrence == 0) {
          chance--;
          return 0;
       }
       else {
          return occurrence;
       }
    }

    //post: construct a Map to record word pattern and set of
    //      words in each word pattern. Return Map to record method.
    private Map<String, Set<String>> returnMap(Set<String> wordRemain,
    char guess) {
       Map<String, Set<String>> map = new TreeMap<>();
       for(String str: wordRemain) {
          String pattern = "";
          for(int i = 0; i < str.length(); i++) {
             if(str.charAt(i) == guess) {
                pattern += guess + " ";
             }
             else {
                pattern += "- ";
             }
          }
          if(!map.containsKey(pattern)) {
             map.put(pattern, new TreeSet<>());
          }
```

```java
            map.get(pattern).add(str);
         }
         return map;
      }

      //post: update the Set with word still available to use
      private void changeWordRemain(Map<String, Set<String>> map) {
         int size = 0;
         for(Set<String> container: map.values()) {
            if(container.size() > size) {
               size = container.size();
               wordRemain = container;
            }
         }
      }

      //post: update the word pattern after a guess and return number of
      //      occurrences of the guessed letter in the new pattern
      private int changeWordFamily(Map<String, Set<String>> map, char guess) {
         int occurrence = 0;
         for(String maxPattern: map.keySet()) {
            if(map.get(maxPattern) == wordRemain) {
               for(int i = 0; i < wordFamily.length(); i++) {
                  if(maxPattern.charAt(i) == guess) {
                     occurrence++;
                     wordFamily = wordFamily.substring(0, i) + guess
                      + wordFamily.substring(i + 1);
                  }
               }
            }
         }
         return occurrence;
      }
   }
```