Lab Report - Exercise 12: Scrabble Cheater Deluxe - Source Code
2016-07-05

Simon Fecke (s0544318)
Julius Deckert (s0545190)
Metawee Langka (s0551423)

Source Code

class Main

```
import java.util.Random;

public abstract class Main {

     // array with the letters in the english version of scrabble
     public static final char[] BAG_OF_SCABBLE_TILES = {
           'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', // 12
           'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',                    // 9
           'i', 'i', 'i', 'i', 'i', 'i', 'i', 'i', 'i',
           'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o',                         // 8
           'n', 'n', 'n', 'n', 'n', 'n',                                        // 6
           'r', 'r', 'r', 'r', 'r', 'r',
           't', 't', 't', 't', 't', 't',
           'l', 'l', 'l', 'l',                                                  // 4
           's', 's', 's', 's',
           'u', 'u', 'u', 'u',
           'd', 'd', 'd', 'd',
           'g', 'g', 'g',
// 3
           'b', 'b', 'c', 'c', 'm', 'm', 'p', 'p', 'f', 'f',               // 2
           'h', 'h', 'v', 'v', 'w', 'w', 'y', 'y',
           'k', 'j', 'x', 'q', 'z',                                             // 1
     };

     public static final boolean DEBUG_MODE = false;

     static Encyclopedia scrabbleEncyclopedia;

     // use small list for debug
     static boolean useSmallList = DEBUG_MODE;

     // default capacity for each dictionary
     static int capacity = 1009;

     // range of word length
     static int minLetters = 2;
     static int maxLetters = 7;
```

```
/********
 * MAIN *
 *******/


public static void main(String[] args) {

     // instantiate dictionary
     if(useSmallList) {
          capacity = 17;
          scrabbleEncyclopedia = new Encyclopedia("scrabble_small.txt",
                     minLetters, maxLetters, capacity);
     } else {
          capacity = 8179;
          scrabbleEncyclopedia = new Encyclopedia("scrabble_full.txt",
                     minLetters, maxLetters, capacity);
     }

     outputStats();

     // Test a known combination
     // takeTurn("babsla");

     // Try 10 random draws
     for(int i=0; i<1; i++) {
          takeTurn(7);
     }
}


/***********
 * METHODS *
 **********/


/*
 * Takes an integer parameter and
 * Returns that many random tiles as string
 * Scrabble rules are ignored (could be 10 times Q)
 */
private static String drawTiles(int numberOfTiles) {
     char[] tiles = new char[numberOfTiles];
     Random rnd = new Random();

     // all possible tiles for english scrabble
     char[] bag = BAG_OF_SCABBLE_TILES;

     // fisher-yates shuffle on bag
```

```
    for(int i=0; i<bag.length-1; i++) {
        int j = rnd.nextInt(bag.length-1);
        if(i!=j) {
            char swap = bag[i];
            bag[i] = bag[j];
            bag[j] = swap;
        }
    }

    // take the first n tiles
    for(int i=0; i<numberOfTiles; i++) {
        tiles[i] = bag[i];
    }

    return String.valueOf(tiles);
}

/*
 * Draws tiles, outputs the draw and then
 * calls the other takeTurn() method with
 * the drawn tiles as parameter.
 */
private static void takeTurn(int numberOfTiles) {
    // generate tiles
    String tiles = drawTiles(numberOfTiles);

    // output tiles
    outputTiles(tiles);

    // then go on with other takeTurn method.
    takeTurn(tiles);
}

/*
 * Takes a list of tiles as string parameter,
 * looks up all possible words for that combination
 * and outputs them to the console.
 */
private static void takeTurn(String tiles) {
    // Look up possible combinations
    String[] possibleWords = scrabbleEncyclopedia.getPossibleWords(tiles);

    // output results
    outputPossibleWords(possibleWords, tiles);
}

/*
 * Outputs Scrabble tiles drawn from the bag
 */
private static void outputTiles(String tiles) {
```

```
          System.out.println();
          System.out.print("The " + tiles.length() + " tiles you drew are: ");
          for(int i=0; i<tiles.length(); i++) {
               System.out.print("'" + tiles.charAt(i) + "' ");
          }
          System.out.println();
     }

     /*
      * Outputs possible words found in the encyclopedia
      */
     private static void outputPossibleWords(String[] possibleWords, String tiles) {
          System.out.println();
          if(possibleWords.length > 0) {
               System.out.println("The possible words for '" + tiles + "' are:");
               for(int i=0; i<possibleWords.length; i++) {
                    System.out.println(possibleWords[i]);
               }
          } else {
               System.out.println("No possible words for '" + tiles + "'.");
          }
          System.out.println();
     }

     /*
      * Outputs some statistics of the encyclopedia
      */
     private static void outputStats() {
          // get statistics
          int valuesStored = scrabbleEncyclopedia.countValuesStored();
          int collisions = scrabbleEncyclopedia.getCollisionCount();
          int emptyBuckets = scrabbleEncyclopedia.countEmptyBuckets();

          // calculate more statistics
          int totalBuckets = capacity * (maxLetters-(minLetters-1));
          int usedBuckets = totalBuckets - emptyBuckets;
          double usedPercent = (usedBuckets / (double)totalBuckets) * 100;
          double collisionPercent = (collisions / (double)valuesStored) * 100;

          // output statistics
          System.out.println("Total buckets: " + totalBuckets);
          System.out.println("Used buckets: " + usedBuckets);
          System.out.println("Empty buckets: " + emptyBuckets);
          System.out.println(usedPercent + "% of capacity is being used.\n");

          System.out.println("Values stored: " + valuesStored);
          System.out.println("Collisions: " + collisions);
          System.out.println(collisionPercent + "% collision rate.\n");
     }
}
```

interface ScrabbleStorage

```
public interface ScrabbleStorage {

      /*
       * Takes a string as parameter and adds it to the storage
       */
      public abstract void add(String str);

      /*
       * Looks up a normalized string, the letter tiles,
       * in the storage and returns all matching words.
       */
      public abstract String[] getPossibleWords(String word);

      /*
       * Returns the number of collisions in the storage
       */
      public abstract int getCollisionCount();

      /*
       * Returns the number of words in all dictionaries
       */
      public abstract int countValuesStored();

}
```

class Permutation

```
import java.util.HashSet;


public class Permutation {

      HashSet<String> permutations;
      String originalWord;
      int minLetters;

      public Permutation(String word, int minLetters) {
            this.originalWord = word;
            this.minLetters = minLetters;
            permutations = new HashSet<String>();
            String normalizedWord = this.normalize(word);
            findPermutations(normalizedWord, permutations);
      }


      /******************
       * PUBLIC METHODS *
       *****************/
```

```
/*
 * Returns the size of the permutations hashset
 */
public int size() {
     return this.permutations.size();
}

/*
 * Returns all values in the permutations hashset as String array
 */
public String[] getValues() {
     String[] returnArray = new String[this.permutations.size()];
     returnArray = this.permutations.toArray(returnArray);

     return returnArray;
}

/*
 * (non-Javadoc)
 * @see java.lang.Object#toString()
 *
 * Override toString() method for convenient output
 */
@Override
public String toString() {
     String returnString = "Permutations of '" + this.originalWord + "':\n";

     // convert HashSet to String-Array
     String[] list = new String[permutations.size()];
     list = this.permutations.toArray(list);

     // append each element to the string
     for(int i=0; i<list.length; i++) {
          returnString += (list[i] + "\n");
     }

     return returnString;
}


/*******************
 * PRIVATE METHODS *
 ******************/

/*
 * Takes a string and a hashset as parameters
 * and recursively puts all possible permutations
 * of that string into the hashset.
 */
```

```
    private void findPermutations(String word, HashSet<String> list) {
        // add word to the list
        list.add(word);

        // run again for every possible (if the word still has more than minimum
letters)
        if(word.length() > minLetters) {
            // remove each character from the string and run again for the
remaining word
            for(int i=0; i<word.length(); i++) {
                // convert to StringBuilder so we can delete chars
                findPermutations(new
StringBuilder(word).deleteCharAt(i).toString(), list);
            }
        }
    }

    /*
     * Takes a word as parameter and returns the normalized String.
     * Sorts the characters alphabetically.
     */
    private String normalize(String word) {
        // convert to char array and convert capital letters to lower case
        char[] letters = word.toLowerCase().toCharArray();
        int length = word.length();

        // use bubble sort -- what performs best at this size < 20?? quick? heap?
        char swap;
        boolean isUnsorted = true;

        while(isUnsorted) {
            isUnsorted = false;
            for(int i=0; i<length-1; i++) {
                if(letters[i] > letters[i+1]) {
                    swap = letters[i];
                    letters[i] = letters[i+1];
                    letters[i+1] = swap;
                    isUnsorted = true;
                }
            }
        }

        return String.valueOf(letters);
    }

}
```

class WordBucket

```java
import java.util.ArrayList;
import java.util.HashSet;


public class WordBucket implements ScrabbleStorage {

    ArrayList<String> bucket;

    public WordBucket() {
        bucket = new ArrayList<String>();
    }


    /*********************
     * INTERFACE METHODS *
     ********************/


    /*
     * Add word to bucket
     */
    public void add(String str) {
        bucket.add(str);
    }

    /*
     * Returns all matches for a string
     */
    public String[] getPossibleWords(String tiles) {
        ArrayList<String> wordsFound = new ArrayList<String>();

        for(String word : bucket) {
            if(this.isPermutation(word, tiles))
                wordsFound.add(word);
        }

        // return the remaining words
        String[] wordsArray = new String[wordsFound.size()];
        return wordsFound.toArray(wordsArray);
    }

    /*
     * Returns the number of collisions in the bucket
     */
    public int getCollisionCount() {
        int collisions = 0;
        // count collisions via HashSet
        HashSet<String> collisionTest = new HashSet<String>();
        for(String s : bucket) {
            // normalize each word and put in hashset
            collisionTest.add(this.normalize(s));
        }
        // count words in hashset - first is not a collision
        collisions += collisionTest.size() - 1;

        // make sure count isn't below zero (e.g. for empty buckets)
        if(collisions < 0)
            collisions = 0;

        return collisions;
    }

    /*
     * Returns the number of words in the bucket
     */
    public int countValuesStored() {
        return bucket.size();
    }


    /******************
     * PUBLIC METHODS *
     *****************/

    /*
     * Return all words as String array
     */
    public String[] toArray() {
        String[] returnArray = new String[bucket.size()];
        returnArray = bucket.toArray(returnArray);

        return returnArray;
    }

    /*
     * Return all words as ArrayList<String>
     */
    public ArrayList<String> asList() {
        return bucket;
    }

    /*
     * Returns a boolean indicating whether the bucket is empty
     */
    public boolean isEmpty() {
        if(bucket.size() == 0)
            return true;

        return false;
    }
```

```
/*
 * (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
     String returnString = "";
     int size = this.bucket.size();

     for(int i=0; i<size; i++) {
          returnString += "'" + this.bucket.get(i) + "'  ";
     }

     return returnString;
}


/*******************
 * PRIVATE METHODS *
 ******************/


/*
 * Takes a word as parameter and returns the normalized String.
 * Sorts the characters alphabetically.
 * MAKE PRIVATE
 */
private String normalize(String word) {
     // convert to char array and convert capital letters to lower case
     char[] letters = word.toLowerCase().toCharArray();
     int length = word.length();

     // use bubble sort -- what performs best at this size < 20?? quick? heap?
     char swap;
     boolean isUnsorted = true;

     while(isUnsorted) {
          isUnsorted = false;
          for(int i=0; i<length-1; i++) {
               if(letters[i] > letters[i+1]) {
                    swap = letters[i];
                    letters[i] = letters[i+1];
                    letters[i+1] = swap;
                    isUnsorted = true;
               }
          }
     }

     return String.valueOf(letters);
```

```
    }

    /*
     * Takes to strings as parameter and checks whether
     * they are permutations of each other by normalizing both.
     */
    private boolean isPermutation(String a, String b) {
        // DEBUG
        if(Main.DEBUG_MODE)
            System.out.println("Testing permutation between '" + a + "' and '" + b
+ "'.");
        return this.normalize(a).equals(this.normalize(b));
    }

}
```

class Dictionary

```
public class Dictionary implements ScrabbleStorage {

    WordBucket[] buckets;
    int capacity;

    public Dictionary(int capacity) {
        // set capacity and valuesStored
        this.capacity = capacity;

        // Initialize array
        buckets = new WordBucket[capacity];

        // Initialize buckets
        for(int i=0; i<capacity; i++) {
            buckets[i] = new WordBucket();
        }
    }


    /********************
     * INTERFACE METHODS *
     ********************/


    /*
     * Add a word to bucket at hash index
     */
    public void add(String str) {
        int hash = this.getHash(str);
        buckets[hash].add(str);

        // DEBUG
        if(Main.DEBUG_MODE)
```

```
                System.out.println("Adding '" + str + "' at hash index " + hash);
    }

    /*
     * Looks up a normalized string, the letter tiles,
     * in the table and returns all matching words.
     */
    public String[] getPossibleWords(String tiles) {
            // make sure the tiles string is normalized
            tiles = this.normalize(tiles);

            // get all entries at hash index
            int index = this.getHash(tiles);
            return this.buckets[index].getPossibleWords(tiles);
    }

    /*
     * Returns the number of collisions in all buckets
     */
    public int getCollisionCount() {
            int collisions = 0;

            for(WordBucket bucket : buckets) {
                collisions += bucket.getCollisionCount();
            }

            return collisions;
    }

    /*
     * Returns the number of words in the dictionary
     */
    public int countValuesStored() {
            int valuesStored = 0;

            for(WordBucket bucket : buckets) {
                valuesStored += bucket.countValuesStored();
            }

            return valuesStored;
    }


    /******************
     * PUBLIC METHODS *
     *****************/


    /*
     * Returns the number of empty buckets in the dictionary
```

```
 */
public int countEmptyBuckets() {
     int emptyBuckets = 0;

     for(WordBucket bucket : buckets) {
          if(bucket.isEmpty())
               emptyBuckets++;
     }

     return emptyBuckets;
}

/*
 * (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
     String returnString = "";

     for(int i=0; i<capacity; i++) {
          returnString += i + ": " + this.buckets[i].toString() + "\n";
     }

     returnString += "\n";

     return returnString;
}


/*******************
 * PRIVATE METHODS *
 ******************/



/*
 * Takes a string as parameter and
 * returns a hash value.
 */
private int getHash(String str) {
     // make sure string is normalized
     str = this.normalize(str);

     // starting values
     int hash = 31;
     int prime = 503;

     for(int i=0; i<str.length(); i++) {
          hash = prime * hash + str.charAt(i);
          hash %= this.capacity;
```

```
		}

		return hash;
	}

	/*
	 * Takes a word as parameter and returns the normalized String.
	 * Sorts the characters alphabetically.
	 * MAKE PRIVATE
	 */
	private String normalize(String word) {
		// convert to char array and convert capital letters to lower case
		char[] letters = word.toLowerCase().toCharArray();
		int length = word.length();

		// use bubble sort -- what performs best at this size < 20?? quick? heap?
		char swap;
		boolean isUnsorted = true;

		while(isUnsorted) {
			isUnsorted = false;
			for(int i=0; i<length-1; i++) {
				if(letters[i] > letters[i+1]) {
					swap = letters[i];
					letters[i] = letters[i+1];
					letters[i+1] = swap;
					isUnsorted = true;
				}
			}
		}

		return String.valueOf(letters);
	}
}
```

class Encyclopedia

```
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.HashSet;


public class Encyclopedia implements ScrabbleStorage {

	Dictionary[] dictionaries;
	int minLetters, maxLetters;

	public Encyclopedia(String filename, int minLetters, int maxLetters, int
capacity) {
		// eg min=2; max=7; ArraySize=6 --> 0-5
```

```
        this.minLetters = minLetters;
        this.maxLetters = maxLetters;
        int numberOfDictionaries = maxLetters-(minLetters-1);
        this.dictionaries = new Dictionary[numberOfDictionaries];

        // initialize each dictionary
        for(int i=0; i<numberOfDictionaries; i++) {
            this.dictionaries[i] = new Dictionary(capacity);
        }

        // read file
        readFile(filename);

        // DEBUG output encyclopedia
        if(Main.DEBUG_MODE)
            System.out.println(this.toString());
    }


    /*********************
     * INTERFACE METHODS *
     ********************/



    /*
     * Takes a string as parameter and adds it to the relevant dictionary
     */
    @Override
    public void add(String str) {
        // add word to dictionary
        this.dictionaries[str.length()-minLetters].add(str);

    }

    /*
     * Looks up a normalized string, the letter tiles,
     * in the table and returns all matching words.
     */
    @Override
    public String[] getPossibleWords(String word) {
        // Hashset for possible words found
        HashSet<String> possibleWords = new HashSet<String>();

        // get permutations from string
        Permutation perm = new Permutation(word, minLetters);
        String[] possibleCombinations = perm.getValues();

        // System.out.println(perm.toString());

        // look for every permutation in corresponding dictionary
    for(String tiles : possibleCombinations) {
        // DEBUG
        if(Main.DEBUG_MODE)
            System.out.println("Tiles: " + tiles);

        // all words found for these tiles in bucket
        String[] wordBucket = this.dictionaries[tiles.length()-minLetters]
                                              .getPossibleWords(tiles);

        // add matches to hashset of possible words
        for(String w : wordBucket) {
            possibleWords.add(w);
        }
    }

    // convert hashset to string array and return
    String[] wordsArray = new String[possibleWords.size()];
    return possibleWords.toArray(wordsArray);
}

/*
 * Returns the number of collisions in all dictionaries
 */
@Override
public int getCollisionCount() {
    int collisions = 0;

    for(Dictionary dict : dictionaries) {
        collisions += dict.getCollisionCount();
    }

    return collisions;
}

/*
 * Returns the number of words in all dictionaries
 */
@Override
public int countValuesStored() {
    int valuesStored = 0;

    for(Dictionary dict : dictionaries) {
        valuesStored += dict.countValuesStored();
    }

    return valuesStored;
}


/*****************
```

```
 * PUBLIC METHODS *
 *****************/


/*
 * Returns the number of empty buckets in all dictionaries
 */
public int countEmptyBuckets() {
    int emptyBuckets = 0;

    for(Dictionary dict : dictionaries) {
        emptyBuckets += dict.countEmptyBuckets();
    }

    return emptyBuckets;
}

/* (non-Javadoc)
 * @see ScrabbleStorage#toString()
 */
@Override
public String toString() {
    String returnString = "";

    for(int i=0; i<this.dictionaries.length; i++) {
        returnString += "Dictionary for " + (minLetters + i) + " words:\n"
                             + this.dictionaries[i].toString();
    }

    return returnString;
}


/******************
 * PRIVATE METHODS *
 ******************/


/*
 * Takes a file name as parameter, and generates a Hashtable
 * from the words in it.
 */
private void readFile(String filename) {
    File myFile = new File(filename);

    try {
        RandomAccessFile raf = new RandomAccessFile(myFile, "r");

        String nextWord;
        while((nextWord = raf.readLine()) != null) {
                int length = nextWord.length();

                if(length <= maxLetters && length >= minLetters) {
                    this.add(nextWord);
                }
            }

            raf.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```