Laboratory 12: Scrabble Cheater Deluxe

(http://people.f4.htw-berlin.de/~weberwu/info2/labs/ExerC.shtml)

1. **Get out your solution to Exercise 11 - or borrow a working one from someone. Remember to give them credit.**

We use our own solution from the lab before.

2. **Write a method `bool isPermutation (String a, String b) {...}` that determines if a and b are permutations. Use this in your output from the cheater so that only permutations of the input string are printed out, and not all of the collisions.**

We write an new method isPermutation() in class HashTable:

```
public boolean isPermutation(String word1, String word2) {
		char[] word1_array = word1.toCharArray();
		Arrays.sort(word1_array);
		char[] word2_array = word2.toCharArray();
		Arrays.sort(word2_array);

		if(Arrays.equals(word1_array, word2_array)) {
			return true;
		} else {
			return false;
		}
	}
```

Two char Arrays get filled with a word, then sorted alphabetically and at least compared to each other. If they are permutations, the method returns true, if not, false.

3. **Adapt your main to generate a random selection of seven letters to start the cheater with.**

We create a new method getRandomWord() in class ScrabbleCheat:

```
//Create a random word with x amount of letters
	private String getRandomWord(int letters) {
		Random r = new Random();
		String randomWord = "";
		//choose x random character between A and Z and append them to the
string
		for(int i = 0; i < letters; i++) {
		char x = (char) (r.nextInt(26) + 65);
			randomWord += "" + x;
		}
		return randomWord;
	}
```

For the chosen amount of letters a for loop creates a random number between 0 – 25, then adds 65 and a letter gets created through ASCII-code. The new letter gets added to the String randomWord.

5. Make a class that upon instantiation with a given String of characters, determines all of the Strings that are substrings in the sense that they only contain letters from the given String, with multiples only up to the number of multiples available. The order of the letters is irrelevant, so this is a bag. For example with 4 letters "JAVA" this would be {"AAJV", "AJV", "AAJ", "AAV", "AA", "AJ", "AV", "JV"}. Don't worry about single letters. Your finger exercise will come in handy here.

We create a new class Substring and need some new methods:

First methd createStringPerms(), which creates all permutations of the word:

```
private void createStringPerms(String beginningString, String endingString) {

		if (endingString.length() <= 1) {

			stringPerms.add(beginningString + endingString);
		}
		else {
			for (int i = 0; i < endingString.length(); i++) {
				String newString = endingString.substring(0, i) +
endingString.substring(i + 1);
				createStringPerms(beginningString +
endingString.charAt(i), newString);
			}
		}
	}
```

This recursive method saves all possible permutations of the word in the HashSet stringPerms (http://www.java2s.com/Tutorial/Java/0100__Class-Definition/RecursivemethodtofindallpermutationsofaString.htm).

Then method createSubString():

```
private void createSubStrings(String sub) {
		char[] perm = sub.toUpperCase().toCharArray();
		Arrays.sort(perm);
		subStrings.add(String.valueOf(perm));

		if((sub.length()-2) <= 0) {
		} else {
			createSubStrings(sub.substring(0, sub.length()-1));
		}
	}
```

Now, for every permutation, we create its substrings. To avoid duplicates we sort everything alphabetically and save it to the HashSet subStrings.

To access the HashSet stringPerms we also add a getter.

6. Now set up the Scrabble Cheater Deluxe: read in 7 letters, split them into collections of 7-, then 6-, then 5-, ... words contained in the input bag of letters. Look up each word in each collection in the corresponding dictionary. If you find something, output it.

We add a new method scrabbleCheatDeluxe() to the class ScrabbleCheat:

```
//Look up all the Substrings of the word and print if there was somthing stored in
the hash table
	public void scrabbleCheatDeluxe(String word) {
		Substring ss = new Substring(word);
		boolean smthnFound = false;

		for(String subString : ss.getSubStrings()) {
			LinkedList<String> wordList;
			if((wordList = dic.lookup(subString)) != null) {
				System.out.println(dic.lookup(subString));
```

```
                        smthnFound = true;
                    }
                }
                if(smthnFound == false) {
                    System.out.println("Sorry, nothing was found for '" + word +
"'.");
                }
        }
```

We create a new object of substring with the word as parameter and use the getter to access the subStrings.

For each substring we use the lookup method to see if there is something.

If something was found, the subSrtrings get printed out.

If nothing was there, the program says `Sorry, nothing was found for “word”.`

Source Code:

Class Dictionary:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;

public class Dictionary {
	HashTable hT;
	private int wordCounter = 0;

	//Initializes the HashTable and fills the Dictionary with the words from the
textfile
	public Dictionary(String textFile, int hashTableSize) throws Exception {
		hT = new HashTable(hashTableSize);
		fillDictionary(textFile);
		System.out.println("The hash table has a size of " + hashTableSize +
".");
		System.out.println("There are " + wordCounter + " words stored in
it.");
	}

	//Fills dictionary's hash table with all the words seperated by a space from
the textfile
	private void fillDictionary(String textFile) throws Exception {
		ArrayList<String> lineList = readFileLines(textFile);

		//For each line from the file
		for(String s : lineList) {
			//Split it in words
			String[] words = s.split(" ");

			//Add each word to the hash table
			for(String word : words) {
				hT.addWord(word);
				wordCounter++;
			}
		}
	}

	//Returns each line from the textfile as an ArrayList
	private ArrayList<String> readFileLines(String textFile) throws IOException
{
		FileReader fr = new FileReader(textFile);
	    BufferedReader br = new BufferedReader(fr);
	    ArrayList<String> lineList = new ArrayList<String>();

	    String thisLine;
	    //As long as the line read is not null, save it to the array list
		while ((thisLine = br.readLine()) != null) {
		  lineList.add(thisLine);
	    }
```

```
            br.close();
            return lineList;
      }

      public LinkedList<String> lookup(String word) {
            int index = hT.getHashValue(word);

            //As long as there's a collision, check the next index (index+1)
            while(hT.isCollision(word, index)) {
                  index++;
                  //If index is bigger than the hash table size, we didn't find
any word list
                  //or if we encounter an empty space
                  if(index > hT.getHashTableSize()-1  || hT.getWordList(index) ==
null) {
                        return null;
                  }
            }
            //Take the word list at the collision-free index from the hash table
and return it
            return hT.getWordList(index);
      }
}
```

Class ScrabbleCheat:

```
import java.util.LinkedList;
import java.util.Random;

public class ScrabbleCheat {
      Dictionary dic;

      public ScrabbleCheat() throws Exception {
            dic = new Dictionary("words.txt", 20000);
            scrabbleCheatDeluxe(getRandomWord(7));
      }

      //Create a random word with x amount of letters
      private String getRandomWord(int letters) {
            Random r = new Random();
            String randomWord = "";
            //choose x random character between A and Z and append them to the
string
            for(int i = 0; i < letters; i++) {
            char x = (char) (r.nextInt(26) + 65);
                  randomWord += "" + x;
            }
            return randomWord;
      }

      //Look up all the Substrings of the word and print if there was somthing
stored in the hash table
      public void scrabbleCheatDeluxe(String word) {
```

```
		Substring ss = new Substring(word);
		boolean smthnFound = false;

		for(String subString : ss.getSubStrings()) {
			LinkedList<String> wordList;
			if((wordList = dic.lookup(subString)) != null) {
				System.out.println(dic.lookup(subString));
				smthnFound = true;
			}
		}
		if(smthnFound == false) {
			System.out.println("Sorry, nothing was found for '" + word +
"'.");
		}
	}

	public static void main(String[] args) throws Exception {
		new ScrabbleCheat();
	}
}
```

Class HashTable:

```
import java.util.Arrays;
import java.util.LinkedList;

public class HashTable {
	private LinkedList<String>[] hashTable;

	//Initializes the hash table with the sized passed in
	@SuppressWarnings("unchecked")
	public HashTable(int size) {
		hashTable = (LinkedList<String>[]) new LinkedList[size];
	}

	//Adds the word passed in to the HashTable at the index according to the
hash value
	public void addWord(String word) throws Exception {
		word = word.toUpperCase();
		int index = getHashValue(word);

		//If theres a collision, check next index
		while(isCollision(word, index)) {
			index++;
			if(index > getHashTableSize()-1) {
				throw new Exception("No space left in hash table.");
			}
		}
		//Is at the index already a list? If no create one
		if(hashTable[index] == null) {
			hashTable[index] = new LinkedList<String>();
		}
		//Store the word in the list
		hashTable[index].add(word);
	}
```

```
	//Returns all the permuations stored at the specific index
	public LinkedList<String> getWordList(int index) {
		return hashTable[index];
	}

	//Returns the hash value of the word passed in
	public int getHashValue(String word) {
		//Seperate each letter of the Word and normalize it to upper cases
		char[] letters = word.toUpperCase().toCharArray();
		//Sort the letters alphabetically
		Arrays.sort(letters);
		//Calculate the hash value. ('A'-1) normalizes the value of the chars
to A = 1, B = 2, C = 3 ..
		int hashValue = 0;
		for (int i = 0; i < word.length(); i++) {
			hashValue += (letters[i] - ('A'-1)) * 17 ^ i;
		}
		//Prevent negative values (Why do they even occur?)
		hashValue = (hashValue < 0)? hashValue*-1:hashValue;
		return hashValue%9973;
	}

	//Returns true if the first word in the list at a specific index is not a
permutation with the specific word
	public boolean isCollision(String word, int index) {
		word = word.toUpperCase();
		//Nothing in here yet? No collision
		if(hashTable[index] == null) {
			return false;
		} else {
			char[] tableWord = hashTable[index].getFirst().toCharArray();
			Arrays.sort(tableWord);
			char[] myWord = word.toCharArray();
			Arrays.sort(myWord);

			//Are both words permutations of each other? No collision!
			if(Arrays.equals(tableWord, myWord)) {
				return false;
			} else {
				//If they are not a permutation of each other:
Collision!
				return true;
			}
		}
	}

	//Returns the size of the hash table
	public int getHashTableSize() {
		return hashTable.length;
	}

	public boolean isPermutation(String word1, String word2) {
		char[] word1_array = word1.toCharArray();
		Arrays.sort(word1_array);
		char[] word2_array = word2.toCharArray();
		Arrays.sort(word2_array);
```

```
		if(Arrays.equals(word1_array, word2_array)) {
			return true;
		} else {
			return false;
		}
	}
}
```

Class Substring:

```
import java.util.Arrays;
import java.util.HashSet;


public class Substring {
	private HashSet<String> subStrings;
	private HashSet<String> stringPerms;

	public Substring(String stringLetters) {
		subStrings = new HashSet<String>();
		stringPerms = new HashSet<String>();

		createStringPerms("", stringLetters);

		for(String perm : stringPerms) {
			createSubStrings(perm);
		}
	}

	public HashSet<String> getSubStrings() {
		return subStrings;
	}

	private void createStringPerms(String beginningString, String endingString)
{

		if (endingString.length() <= 1) {

			stringPerms.add(beginningString + endingString);
		}
		else {
			for (int i = 0; i < endingString.length(); i++) {
				String newString = endingString.substring(0, i) +
endingString.substring(i + 1);
				createStringPerms(beginningString +
endingString.charAt(i), newString);
			}
		}
	}

	private void createSubStrings(String sub) {
		char[] perm = sub.toUpperCase().toCharArray();
		Arrays.sort(perm);
		subStrings.add(String.valueOf(perm));

		if((sub.length()-2) <= 0) {
```

```
		} else {
			createSubStrings(sub.substring(0, sub.length()-1));
		}
	}
}
```