Lab Report

Exercise 11: Scrabble Cheater Basic Edition

Name		Titel des Kurses	Datum
Juri Wiechmann	57108 5	Prof. Dr. Weber-Wulff Info 2 Group 2	16.01.2020
Bartholomäus Berresheim	56862 4		

Index

Bartholomäus

Code

Introduction

Pre-lab

1. Review the rules of Scrabble, if you have never played it before.

Since we had played scrabble before, we didn’t need to look the rules up.

2. What would the exact data structure be for a hash table that stores Strings and chains the collisions?
Either we have for each index of the hash table, a String in which we save the first value that we want to save at that index, and if there ever comes a collision for that index, we create a LinkedList in which we save the collision. Or from the very beginning, we save every String in the LinkedList.

3. What would a normalization function for words look like for a hash? That is, "JAVA" and "VAJA" are permutations, what would a normalized permutation look like?
A way to normalise the permutations, so that we would get the same value for JAVA and VAJA, would be to take the ASCII value of their chars or their place in the Alphabet, and multiply/add the values to each other.
Ex: JAVA = 74 x 65 x 86 x 65 = 26 887 900
VAJA = 86 x 65 x 74 x 65 = 26 887 900
The code for this function would look like this:

```
public int normalisation (String input)
int result = 1;
input.removeAll(“\\s+”, ””)
{
	for (int i = 0; i< input.length(); i++)
	{
		int a = input.charAt(i)
		result = result * a;
	}
	return result;
}
```

4. How do you determine if two strings are permutations of each other?
You can determine if two String are permutations of each other by looking at the chars and their amount contained in both Strings. If both Strings contain the same amount of the same chars, then they are permutations of each other.

5. Review the construction of a hash function. Note that you will need prime numbers. Does your isPrime method work? If not, fix it now.

Since we had hash functions in class, there was no need to review them, and our isPrime() method was working, so there was no need to fix it.

6. Can you find lists of valid words for Scrabble in English online? Are there perhaps any sorted by number of letters in the word? Or maybe one file for each word size? Note down the URLs!

For this task, we found the "https://www.wordfind.com/scrabble-word-list/" and the "http://www.poslarchive.com/math/scrabble/lists/index.html" websites. They contain multiples word lists, including: 2 letter words, 3 letter words, 4 letter words, etc.

Lab Assignments

1. Write a dictionary class that upon instantiation reads in a file of words and creates a hash table for storing them. Use chaining of collisions in your hash table. How many entries does your table have? How many collisions were there? What is the longest chain in your hash table? It might be useful to implement some statistical methods in order to see if your hash table is "okay". Can you fix your hash function in order to only have chains of 16 or less?

For this task we first created an interface HashTable, in which we defined some methods for an HashTable data type.

```
public interface HashTable {
    public void resize();
    public void add(String item);
    public boolean contains(String item);
    public void reset();
    public String toString();
    public void print();
}
```

We then continued by creating a class MyHashTable in which we implemented the previously mentioned interface. In the class, we first added a constructor, in which we set the value of an int field "M" to 24593. We then set the value of M to a calling of the getNextPrime(). getNextPrime() is a method that takes M and

using the isPrime() method it looks for the next Prime number bigger than M, which it then returns.

```
private int getNextPrime() {
    for(int i = M;true;i++) {
        if(UM.isPrime((long) i)) {
            return i;
        }
    }
}
```

We then continue, by setting the value of the int field "listsSize" to 16, and the LinkedList field "item" to a new LinkedList of size M. We then add a LinkedList at every index of "item" using a for loop. We had to do this because otherwise they wouldn't be initialized correctly.

```
public class MyHashTable<Item> implements HashTable {

    int M;
    int listsSize;
    LinkedList<String>[] items;
    int maxSteps;

    public MyHashTable() {
        M = 24593;
        M = getNextPrime();
        listsSize = 16;
        items = new LinkedList[ M];

        for(int i = 0; i<items.length;i++) {
            items[i]= new LinkedList<String>();
        }
    }
```

After that we added a second constructor, this one takes an integer M as input. It works exactly like the previous one but in this one we set the value of "M" to the inputted M.

```
public MyHashTable(int M) {
    this.M = M;
    this.M = getNextPrime();
    items = new LinkedList[M];

    for(int i = 0; i<items.length;i++) {
        items[i]= new LinkedList<String>();
    }
}
```

After that, we added the resize() method, in it we print out a message to the console, followed by the creation of a new HashTable "newHashTable" with the size of M*2. After that we add every List in "items" to "newHashTable".

```java
@Override
public void resize() {
    System.out.println("resizing");
    MyHashTable<String> newHashTable = new MyHashTable<String>(M*2);

    for(LinkedList<String> List : items) {
        for(String item : List) {
            newHashTable.add(item);
        }
    }
}
```

Following that, we created a method called add(), that gets a String "item" as input. In it we first created a char array "stringAsChars" and set its value to the Uppercase CharArray conversion of "item". After that we added 4 variable: 2 ints "i" and "value" with both having 0 as value, and 2 BigIntegers "BigIndex" having BigInteger.ZERO as value and "digitvalue". We then created a for each char of "stringAsChars" loop. In it we set the value of "value" to the ASCII value of "c". We then proceed by setting the value of "digitValue" to the one from "value. We then set the value of a long "multiplicator" to the result of (27^i). After that we use "multiplicator" and multiply it with the value of "digitValue", the result of that is then added to the value of "BigIndex". And finally we add 1 to "i".

```java
public void add(String item) {
    char[] stringAsChars = item.toUpperCase().toCharArray();

    int i = 0;
    int value = 0;

    BigInteger BigIndex = BigInteger.ZERO;
    BigInteger digitValue;

    for(char c : stringAsChars){
        value = c;
        digitValue = new BigInteger(value+"");
        long multiplicator = (long) Math.pow(20, i);
        BigIndex = BigIndex.add(digitValue.multiply(new BigInteger(multiplicator+"")));
        i++;
    }
```

After the loop we set the value of "BigIndex" to its previous value Mod M. Afterwards we set the value of "value" to the int value of "Bigindex". We then proceed by checking whether the list in "items" at the index equal to "value" is bigger than "listsSize"-1, since we want to keep the lists to a max size of 16. If true, we set the value of "value" to a calling of the getNewIndex() method. After the if statement we add the Uppercase version of "item" to "items" at the index of "value.

```
BigIndex = BigIndex.mod(new BigInteger(M+""));
value = BigIndex.intValue();

//pro
if(items[value].size()>listsSize-1) {
    value = getNewIndex(value,1);
}
items[value].add(item.toUpperCase());
```

The getNewIndex() method gets 2 inputs: an int “value” and an int “steps”. In it we first set the value of an int “change” to the result of “steps”^2. After that we check whether “value” is even. If true, we check whether the list in “items” bordering the one, that has an index of “value”, has a size smaller than “listsSize”, if true, we return value-change. If not, we call the getNewIndex() method with “value-change” and “steps+1” as inputs. If value is not even, we do the same but instead of subtraction “change” from “value”, we add it. We did this to spread the inputs in “items, in a better way.

```
private int getNewIndex(int value, int step) {

    int change = (int)Math.pow(step, 2);
    //even numbers change to left
    if(value % 2 == 0) {
        if(items[bordered(value-change)].size()<listsSize){
            return bordered(value-change);
        }
        else {
            return bordered(getNewIndex(value-change, step+1));
        }
    }
    //odd numbers change to right
    else{
        if(items[bordered(value+change)].size()<listsSize){
            return bordered(value+change);
        }
        else {
            return bordered(getNewIndex(value+change, step+1));
        }
    }
}
```

In the getNewIndex() method we called a method called bordered(),that returns the index of “value-change”. In the method, we make sure that “value-change” doesn’t return a negative index or one that is bigger than the size of “items”.

```
private int bordered(int index) {
    //already in border
    if(index<items.length && index > -1)
        return index;
    else {
        //if its bigger
        if(index >items.length-1) {

            return bordered(index-items.length);
        }
        //if its lower
        else {
            return bordered(items.length+index);
        }
    }
}
```

After finishing that, we continued by writing some useful methods. The first one is the reset() method, it sets “items” to a new LinkedList of size “M”.

```
@Override
public void reset() {
    items = new LinkedList[M];
}
```

The next method, we added was the toString() method, in it we set the value of a String “outcome” to “”, followed by a for each List in “items” loop. In the loop, we have a for each “item” in “List” loop, in which we add “item” and “ \n” to “outcome”. After the loops we return “outcome”.

```
@Override
public String toString() {
    String outcome = "";

    for(LinkedList<String> List : items) {
        for(String item : List) {
            outcome = outcome + item + "\n";
        }
    }
    return outcome;
}
```

Afterwards, we added a print() method, in which we print the result of the toString() method.

```
@Override
public void print() {
    System.out.println(toString());
}
```

We then continued by adding a method called “printListSize”. In it we check every List in “items” and we return their size if it is bigger than 0.

```
public void printListsSize() {
    for(LinkedList<String> list : items) {
        if(list.size()>0)
            System.out.println(list.size());
    }
}
```

And lastly, we added a method called gerEmptyLists(), in which we first count the number of every empty list in "items", after which we turn it into a percentage value.

```
public double getEmptyLists() {
    int i = 0;
    for(LinkedList<String> list : items) {
        if(list.size() == 0)
            i++;
    }
    double percent= ((double)i/items.length)*100;
    percent = Math.round(percent*100);
    return percent/100;
}
```

For the reading in of a file, we reused a method from a previous lab. It gets a filePath as input, to which we add a FileReader, to which in turn we add a BufferedReader. Then in a while loop, we add the Lines of the File to a String "output", which gets returned after the loop.

```
public static String FileToString(String filePath) throws IOException {
    FileReader fr = new FileReader(filePath);
    BufferedReader br = new BufferedReader(fr);

    String outcome = "";

    while(br.ready()) {
        outcome += br.readLine() + " ";
    }


    System.out.println("File loader complete");
    return outcome;
}
```

Now that everything is set in place, we created a Class with a static main() method.

In it we first created a new Object of type MyHashTable "dictionary". We then print out the size of "dictionary", after which we set the value of a String "filePath" to the file path of our dictionary. Afterwards we added a for each "word" in our file, in which we add every word to "dictionary.

```
public static void main(String[] args) throws Exception {

    //Filling dictionary
    MyHashTable<String> dictionary = new MyHashTable<String>();
    System.out.println("TableSize: " + dictionary.M);
    String filePath = "S:\\the real dictionary.txt";


    for(String word : UM.FileToString(filePath).split("\\s+")) {
        dictionary.add(word);
    }
```

After which, we add a calling of the printListsSize() and of the getEmptyLists() methods.

Now Once executed, we get :

```
TableSize: 24593
File loader complete
Amount of empty lists: 11.26%
1
3
2
2
2
1
2
4
4
1
1
```

(The output on the console goes on for 24 500 more messages)

As we can see 11.26% of the Lists in our HashTable are empty. And while scrolling through the console, we can see that most aren't even half full. After making some modifications to the getEmptyLists() method, so that it outputs how many Lists are less than half full, and how many are less than a quarter full. And we get the following results:

```
Amount of lists that are less than half full: 99.8%

Amount of lists that are less than a quarter full: 82.34%
```

From that, we can see that our inputted File is distributed in a way that they don't get clustered in a certain point.

2 & 3

You will need to have a lookup method in your class that takes a word (i.e. a String) and returns an array of Strings corresponding to all the words at the hash location, if any. You may need to normalize the word to look up, depending on your hash function.

Now make the basic Scrabble cheater: construct a 7-letter-word hash dictionary, set a String to 7 letters, and output the array of Strings found that might be permutations of these 7 letters. Your users can check if there is a permutation to be found. Or you can implement isPermutation and only output the ones that are permutations if you are bored.

For this task, we first created a method called contains() that gets a String "item" as input. The first part of it, in which we calculate the index for the input, is identical to the first part of the add() method. After that we set the value of a LinkedList "MyLinkedList" to the LinkedList in "items" at the index of "value". We then check whether "MyLinkedList" contains "item", if yes we return true. If not, we check if the size of "MyLinkedList" is bigger then "listsSize"-1, meaning we check if there was a collision. If its false, we return false, and if true we return a calling of the findWord() method, that has "item" "value and "1" as input.

```
@Override
public boolean contains(String item) {
    char[] stringAsChars = item.toUpperCase().toCharArray();
    LinkedList<String> MyLinkedList = null;
    int i = 0;
    int value;

    BigInteger BigIndex = BigInteger.ZERO;
    BigInteger digitValue;
    for(char c : stringAsChars){
        value = c;
        digitValue = new BigInteger(value+"");
        long multiplicater = (long) Math.pow(20, i);
        BigIndex = BigIndex.add(digitValue.multiply(new BigInteger(multiplicater+"")));
        i++;
    }
    BigIndex = BigIndex.mod(new BigInteger(M+""));
    value = BigIndex.intValue();
    MyLinkedList = items[value];

    if(MyLinkedList.contains(item.toUpperCase())) {
        return true;
    }
    else {
        if(items[value].size()>listsSize-1) {
            return findWord(item.toUpperCase(),value,1);
        }
        return false;
    }
}
```

The findWord() method works almost exactly like the getNewIndex() method. It gets 3 inputs: a String “word”, an int “value” and an int “step”. It then calculates a new index and looks if the List at the new index contains “word” and if not whether that list is full. If it contains the word it returns true, if not and the List is not full it returns false, and if it’s neither of those cases, it returns a calling of the findWord() method with the updated value and a higher amount of steps.

```
private boolean findWord(String word, int value, int step) {
    int change = (int)Math.pow(step, 2);
    //even numbers change to left
    if(value % 2 == 0) {
        value = bordered(value-change);
        if(items[value].contains(word)) {
            return true;
        }
        else {
            if(items[value].size()<listsSize) {
                return false;
            }
            else {
                return findWord(word, value, step+1);
            }
        }
    }
    //odd numbers change to right
    else {
        value = bordered(value+change);
        if(items[value].contains(word)) {
            return true;
        }
        else {
            if(items[value].size()<listsSize) {
                return false;
            }
            else {
                return findWord(word, value, step+1);
            }
        }
    }
}
```

And finally we added a method called lookup(), that gets a String “bench” as input. In it we first set the value of a Set “permute” to the calling of the method permute(), having “bench” as input.

```
public ArrayList<String> lookup(String bench) {
    Set<String> permute = UM.permute(bench);
```

The method permute() is something we found at "https://stackoverflow.com/questions/9666903/every-combination-of-character-array". It is a method that takes a String as input. It then checks whether the String only contains one char, and if true it adds the char to a Set "set", which it then returns.If false, it continues with a for loop, in which it seperates the input into 2 substrings, so that neither contain the char at index "i". It then adds the 2 substrings back together and using a for each loop and recursion, it finds all the permutations for the remain chars. After which it returns "set".

```
//https://stackoverflow.com/questions/9666903/every-combination-of-character-array
public static Set<String> permute(String chars){
    // Use sets to eliminate semantic duplicates (aab is still aab even if you switch the two 'a's)
    // Switch to HashSet for better performance
    Set<String> set = new TreeSet<String>();

    // Termination condition: only 1 permutation for a string of length 1
    if (chars.length() == 1){
      set.add(chars);
    }
    else{
        // Give each character a chance to be the first in the permuted string
        for (int i=0; i<chars.length(); i++){
            // Remove the character at index i from the string
            String pre = chars.substring(0, i);
            String post = chars.substring(i+1);
            String remaining = pre+post;

            // Recurse to find all the permutations of the remaining chars
            for (String permutation : permute(remaining)){
                // Concatenate the first character with the permutations of the remaining chars
                set.add(chars.charAt(i) + permutation);
            }
        }
    }
    return set;
}
```

After that we create a new ArrayList "existsPermutation". Followed by a for each String "per" in "permute" loop, in which we call the contains() method using "per" and if it returns true, we add "per" to "existsPermute". After the loop, we return "existsPermute".

```
public ArrayList<String> lookup(String bench) {
    Set<String> permute = UM.permute(bench);
    ArrayList<String> existsPermute = new ArrayList<String>();
    for(String per : permute) {
        if(contains(per))
            existsPermute.add(per.toUpperCase());
    }
    return existsPermute;
```

Now that everything is set in place, we go back to the class where we tested our previous methods, and added a String “bench” in which we set some random letters as value, followed by a for each “permute” in dictionary.lookup(bench), in which we print every “permute”.

```
//Search for words with letters (sequence irrelevant)
String bench = "siufhwl";
for(String permute : dictionary.lookup(bench)) {
    System.out.println(permute);
}
```

Once execute, we get:

```
TableSize: 24593
File loader complete
WISHFUL
Amount of empty lists: 11.26%
```

For `String bench = "eetstrl";`

We got:

```
TableSize: 24593
File loader complete
LETTERS
SETTLER
STERLET
TRESTLE
Amount of empty lists: 11.26%
```

Since our dictionary contains words ranging from 1-7 letter words, we also tried different sized ones.

For `String bench = "aelst";`
We got:

```
TableSize: 24593
File loader complete
LEAST
SETAL
SLATE
STALE
STEAL
STELA
TAELS
TALES
TEALS
TESLA
Amount of empty lists: 11.26%
```

Or for : `String bench = "tide";`

We get:

```
TableSize: 24593
File loader complete
DIET
DITE
EDIT
TIDE
TIED
Amount of empty lists: 11.26%
```

We can thus conclude that our code is working as intended.

Reflections

Juri:

This week's lab was comparable easy to the last one. We worked again with LinkedLists and Set like we learned in the last lab. Also the slides and the lecture was very helpful to understand the mechanism behind the HashTable data structure.

Bartholomäus:

I'm happy that we managed to finish this week's lab much earlier than the last one, since we have to spend a lot of our free time for studying now. As usual there were some things that didn't go smoothly from the get go. One of them was assembling the dictionary, we had found multiple lists online but most of them had problems, such as removing the white spaces between the words once copied, or only displaying a max 500 words per list without giving us a way of accessing the rest, having other information next to the words that gets copied with the words and so on… And as usual Juri was a big help in doing the lab.

Code

```
package ubung11;

public interface HashTable {
	public void resize();
	public void add(String item);
	public boolean contains(String item);
	public void reset();
	public String toString();
	public void print();
}
package ubung11;
```

```
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Set;

public class MyHashTable<Item> implements HashTable {

	int M;
	int listsSize;
	LinkedList<String>[] items;
	int maxSteps;

	public MyHashTable() {
		M = 24593;
		M = getNextPrime();
		listsSize = 16;
		items = new LinkedList[ M];

		for(int i = 0; i<items.length;i++) {
			items[i]= new LinkedList<String>();
		}
	}
	public MyHashTable(int M) {
		this.M = M;
		this.M = getNextPrime();
		items = new LinkedList[M];

		for(int i = 0; i<items.length;i++) {
			items[i]= new LinkedList<String>();
		}
	}
	@Override
	public void resize() {
		System.out.println("resizing");
		MyHashTable<String> newHashTable = new
MyHashTable<String>(M*2);

		for(LinkedList<String> List : items) {
			for(String item : List) {
				newHashTable.add(item);
			}
		}

	}
	@Override
```

```
	public void add(String item) {
		char[] stringAsChars = item.toUpperCase().toCharArray();

		int i = 0;
		int value = 0;

		BigInteger BigIndex = BigInteger.ZERO;
		BigInteger digitValue;

		for(char c : stringAsChars){
			value = c;
			digitValue = new BigInteger(value+"");
			long multiplicator = (long) Math.pow(20, i);
			BigIndex = BigIndex.add(digitValue.multiply(new
BigInteger(multiplicator+"")));
			i++;
		}
		BigIndex = BigIndex.mod(new BigInteger(M+""));
		value = BigIndex.intValue();

		//pro
		if(items[value].size()>listsSize-1) {
			value = getNewIndex(value,1);
		}
		items[value].add(item.toUpperCase());

		//resize if we got less then 1/3 free lists
		/*
		if(33>getEmptyLists()) {
			resize();
		}
		*/
	}
	@Override
	public boolean contains(String item) {
		char[] stringAsChars = item.toUpperCase().toCharArray();
		LinkedList<String> MyLinkedList = null;
		int i = 0;
		int value;

		BigInteger BigIndex = BigInteger.ZERO;
		BigInteger digitValue;
		for(char c : stringAsChars){
			value = c;
			digitValue = new BigInteger(value+"");
```

```
                long multiplicater = (long) Math.pow(20, i);
                BigIndex = BigIndex.add(digitValue.multiply(new
BigInteger(multiplicater+"")));
                i++;
            }
            BigIndex = BigIndex.mod(new BigInteger(M+""));
            value = BigIndex.intValue();
            MyLinkedList = items[value];

            if(MyLinkedList.contains(item.toUpperCase())) {
                return true;
            }
            else {
                if(items[value].size()>listsSize-1) {
                    return findWord(item.toUpperCase(),value,1);
                }
                return false;
            }
        }
        @Override
        public void reset() {
            items = new LinkedList[M];
        }
        private int getNextPrime() {
            for(int i = M;true;i++) {
                if(UM.isPrime((long) i)) {
                    return i;
                }
            }
        }
        @Override
        public void print() {
            System.out.println(toString());
        }
        @Override
        public String toString() {
            String outcome = "";

            for(LinkedList<String> List : items) {
                for(String item : List) {
                    outcome = outcome + item + "\n";
                }
            }
            return outcome;
        }
```

```
	public ArrayList<String> lookup(String bench) {
		Set<String> permute = UM.permute(bench);
		ArrayList<String> existsPermute = new ArrayList<String>();
		for(String per : permute) {
			if(contains(per))
				existsPermute.add(per.toUpperCase());
		}
		return existsPermute;
	}
	public void printListsSize() {
		for(LinkedList<String> list : items) {
			if(list.size()>0)
				System.out.println(list.size());
		}
	}
	private int getNewIndex(int value, int step) {

		int change = (int)Math.pow(step, 2);
		//even numbers change to left
		if(value % 2 == 0) {
			if(items[bordered(value-change)].size()<listsSize){
				return bordered(value-change);
			}
			else {
				return bordered(getNewIndex(value-change,
step+1));
			}
		}
		//odd numbers change to right
		else{
			if(items[bordered(value+change)].size()<listsSize){
				return bordered(value+change);
			}
			else {
				return bordered(getNewIndex(value+change,
step+1));
			}
		}
	}
	//gets an int thats round about the array
	private int bordered(int index) {
		//already in border
		if(index<items.length && index > -1)
			return index;
		else {
```

```
                //if its bigger
                if(index >items.length-1) {

                        return bordered(index-items.length);
                }
                //if its lower
                else {
                        return bordered(items.length+index);
                }
        }
}
private boolean findWord(String word, int value, int step) {
        int change = (int)Math.pow(step, 2);
        //even numbers change to left
        if(value % 2 == 0) {
                value = bordered(value-change);
                if(items[value].contains(word)) {
                        return true;
                }
                else {
                        if(items[value].size()<listsSize) {
                                return false;
                        }
                        else {
                                return findWord(word, value, step+1);
                        }
                }
        }
        //odd numbers change to right
        else {
                value = bordered(value+change);
                if(items[value].contains(word)) {
                        return true;
                }
                else {
                        if(items[value].size()<listsSize) {
                                return false;
                        }
                        else {
                                return findWord(word, value, step+1);
                        }
                }
        }
}
public double getEmptyLists() {
```

```
		int i = 0;
		for(LinkedList<String> list : items) {
			if(list.size() == 0)
				i++;
		}
		double percent= ((double)i/items.length)*100;
		percent = Math.round(percent*100);
		return percent/100;
	}
}
package ubung11;

public class ScrabbleWordFinder {


	public static void main(String[] args) throws Exception {

		//Filling dictionary
		MyHashTable<String> dictionary = new MyHashTable<String>();
		System.out.println("TableSize: " + dictionary.M);
		String filePath = "S:\\the real dictionary.txt";



		for(String word : UM.FileToString(filePath).split("\\s+")) {
			dictionary.add(word);
		}

		//Search for words with letters (sequence irrelevant)
		String bench = "eetstrl";
		for(String permute : dictionary.lookup(bench)) {
			System.out.println(permute);
		}

		System.out.println("Amount of empty lists: "
+dictionary.getEmptyLists() + "%");
		//dictionary.printListsSize();
	}
}

package ubung11;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
```

```java
import java.math.BigInteger;
import java.util.Set;
import java.util.TreeSet;

import ubung9.Point;

public interface UM {

	public static String deleteSpaces(String s) {

		return s.replace("\\s+", "");
	}

	public static boolean StringisNumber(String number) {

		if(number.startsWith("."))
			return false;

		char[] cNumber = number.toCharArray();

		int i = 0;
		try {
			if(cNumber[0] == '-' && CharisNumberHEX(cNumber[1]))
				i = 1;
		}
		catch (Exception e) {}

		for (; i < cNumber.length ; i++) {
			if(!(CharisNumberHEX(cNumber[i]) || cNumber[i] == '.'))
				return false;
		}
		return true;
	}
	public static boolean CharisNumberHEX(char number) {
		if( number > 47 && number < 58 ||
			number > 64 && number < 71)
			return true;
		else
			return false;
	}
	public static String setSpacesEachToken(String S) {
		S = deleteSpaces(S);
		char [] S_Array = S.toCharArray();

		//If the input String has 2 dots in a row its wrong Tested here.
```

```
		int count = 0;
		for(char E : S_Array) {
			if(E == '.')
				count++;
			else
				count = 0;
			if(count >1)
				return null;
		}

		String token = "";
		S = "";
		for(int i = 0; i< S_Array.length;i++) {
			token = "";

			//negativ numbers
			if(	S_Array[i] == '-') {
				token += S_Array[i];

				try {
					//if its an neg followed by a number
					if(CharisNumberHEX(S_Array[i+1])){
						//If - is the first digit:
						if(i==0) {

while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1] == '.') {
								i++;
								token += S_Array[i];
							}
						}
						else {
							//if the - is not at the first digit
							if(CharisOperator(S_Array[i-1]))
{

while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1] == '.') {
									i++;
									token +=
S_Array[i];
							}
						}
					}
					//if the neg is followed by an (
					else if(S_Array[i+1] == '(') {
```

```
                                        i++;
                                        token += S_Array[i];
                                }
                        }
                        catch(Exception e) {}
                        S += token + " ";
                        continue;
                }
                //operatoren
                if( i > 0  && CharisOperator(S_Array[i])) {
                        token += S_Array[i];
                        S += token + " ";
                        continue;
                }
                //positive Zahlen
                else if(CharisNumberHEX(S_Array[i])){
                        token = Character.toString(S_Array[i]);
                        try {
                                while(CharisNumberHEX(S_Array[i+1]) ||
S_Array[i+1] == '.') {
                                        i++;
                                        token += S_Array[i];
                                }
                        }
                        catch(Exception e) {}
                        S += token + " ";
                        continue;
                }
                else {
                        //If input is wrong.
                        return null;
                }
        }


        return S.stripTrailing();
    }
    public static boolean CharisToken(char c) {
        if(CharisNumberHEX(c) || CharisOperator(c)) {
                return true;
        }
        else
                return false;
    }
    public static boolean CharisOperator(char c) {
```

```
		if( c == '+' ||
			c == '-' ||
			c == 'x' ||
			c == '*' ||
			c == '/' ||
			c == '^' ||
			c == '(' ||
			c == ')' ||
			c == '=')
			return true;
		else
			return false;
	}
	public static String getSep(String input) {
		//if you want too really seaching for an sepperator u need a more complicated way, this is the bugded version
		if(input.contains("/"))
			return  "/";
		else if(input.contains("-"))
			return  "-";
		else if(input.contains("."))
			return  ".";
		else if(input.contains("_"))
			return  "_";
		return null;
	}
	public static int twoPointsDistance(int x1, int y1, int x2, int y2) {

		return (int) Math.sqrt((y2 - y1) * (y2 - y1) + (x2 - x1) * (x2 - x1));
	}
	public static Point getMidpoint(Point p1, Point p2) {

		return new Point((p1.getX()+p2.getX())/2,(p1.getY()+p2.getY())/2);
	}
	//doesnt work right now
	public static boolean isPrime (BigInteger n) {

		BigInteger THREE = BigInteger.TWO.add(BigInteger.ONE);

		if(0>n.compareTo(BigInteger.TWO))

			return false;
```

```
		if(0 == n.compareTo(BigInteger.TWO) || 0 ==
n.compareTo(THREE))

			return true;

		//https://www.tutorialspoint.com/java/math/biginteger_mod.htm
		boolean first =
n.mod(BigInteger.TWO).compareTo(BigInteger.ZERO) == 0;
		boolean second = n.mod(THREE).compareTo(BigInteger.ZERO)
== 0;
		if(first || second)
			return false;


//https://www.geeksforgeeks.org/biginteger-sqrt-method-in-java/
		BigInteger sqrtN = n.sqrt().add(BigInteger.ONE);

		BigInteger SIX = new BigInteger("6");
		BigInteger negativOne = new BigInteger("-1");

		for(BigInteger i = SIX; sqrtN.compareTo(i) != -1; i = i.add(SIX))
{
			first =
n.mod(SIX.add(negativOne)).compareTo(BigInteger.ZERO) == 0;
			second =
n.mod(SIX.add(BigInteger.ONE)).compareTo(BigInteger.ZERO) == 0;
			if(first||second ) {
				return false;
			}
		}
		return true;
	}
	public static boolean isPrime (long n) {
		if (n<0) {
			return false;}
		for (long i =2; i<n; i++){
			if (n%i == 0)
			{
				return false;
			}
		}
		return true;
	}
	public static BigInteger pow(BigInteger base, BigInteger exponent) {
		  BigInteger result = BigInteger.ONE;
```

```
            while (exponent.signum() > 0) {
              if (exponent.testBit(0)) result = result.multiply(base);
              base = base.multiply(base);
              exponent = exponent.shiftRight(1);
            }
            return result;
    }
    public static String FileToString(String filePath) throws IOException {
            FileReader fr = new FileReader(filePath);
            BufferedReader br = new BufferedReader(fr);

            String outcome = "";

            while(br.ready()) {
                    outcome += br.readLine() + " ";
            }


            System.out.println("File loader complete");
            return outcome;
    }


//https://stackoverflow.com/questions/9666903/every-combination-of-character-array
    public static Set<String> permute(String chars){
        // Use sets to eliminate semantic duplicates (aab is still aab even if you switch the two 'a's)
        // Switch to HashSet for better performance
        Set<String> set = new TreeSet<String>();

        // Termination condition: only 1 permutation for a string of length 1
        if (chars.length() == 1){
          set.add(chars);
        }
        else{
            // Give each character a chance to be the first in the permuted string
            for (int i=0; i<chars.length(); i++){
                    // Remove the character at index i from the string
                    String pre = chars.substring(0, i);
                    String post = chars.substring(i+1);
                    String remaining = pre+post;

                    // Recurse to find all the permutations of the remaining
```

```
chars
                    for (String permutation : permute(remaining)){
                        // Concatenate the first character with the
permutations of the remaining chars
                        set.add(chars.charAt(i) + permutation);
                    }
                }
            }
            return set;
        }
}
```