Finding all permutations of a String in a Java Program is a tricky question and asked many times in interviews.

http://k2code.blogspot.bg/2011/09/permutation-of-string-in-java-efficient.html

All permutations of a string

Problem

Write a method to compute all permutations of a string.

Example
For a string of length n, there are n! permutations.

INPUT: "abc"
OUTPUT: "abc" "acb" "bac" "bca" "cab" "cba"

So, we have 3! = 6 items for string abc.

Solution

There are several ways to do this. Common methods use recursion, memorization, or dynamic programming.

The basic idea is that you produce a list of all strings of length 1, then in each iteration, for all strings produced in the last iteration, add that string concatenated with each character in the string individually. (the variable index in the code below keeps track of the start of the last and the next iteration)

Lets split up the problem in 2 cases - when duplicates are not allowed and when allowed.

Case 1 : If String has "NO" duplicate

Method 1 - Recursion
Answer: Here is a recursive solution to print all the permutations of a string.

The idea is to keep the first character constant and generate permutations with the rest. Then keep the first two characters constant and generate permutations with the rest until you are out of characters.
So for string "abc", the idea is that the permutations of string abc are a + permutations of string bc, b + permutations of string ac and so on.

The following piece of a code is a very efficient use of recursion to find the possible permutation of a

string.

Caution : However, this solution does not take care of duplicates. It is assumed that there are no duplicates in the string.

```
?
1
2   public class Permutations {
3
4       public static void permutate(String s) { permutateHelper("", s); }
5   //Helper function
6       private static void permutateHelper(String prefix, String rest) {
7           int N = rest.length();
8           if (N == 0)
9               System.out.println(prefix);
10          else {
11              for(int i = 0; i < N; i++){
12                perm1(prefix + rest.charAt(i), rest.substring(0, i)
13                                              + rest.substring(i+1, N));
14              }
15          }
16      }
17
18      public static void main(String[] args) {
19         String alphabet = "Testt";
20         String elements = alphabet.substring(0, alphabet.length());
21         perm1(elements);
22      }
23
```

Dry running the code

```
---ABC
A---BC
AB---C
ABC---
------------------------------ABC
AC---B
ACB---
------------------------------ACB
B---AC
BA---C
BAC---
------------------------------BAC
BC---A
BCA---
------------------------------BCA
C---AB
CA---B
CAB---
------------------------------CAB
CB---A
CBA---
------------------------------CBA
```

Method 1b) Recursion and Backtracking

This will also not work for duplicates. I left out the implementation of the swap method since that implementation is not important here.

```
?
1    //actual helper function
2    void permutateHelper( char[] str, int index )
3    {
4        int i = 0;
5        if( index == strlen(str) )
6        { // We have a permutation so print it
7            printf(str);
8            return;
9        }
        for( i = index; i < strlen(str); i++ )
```

```
10          {
11              swap( str[index], str[i] ); // It doesn't matter how you swap.
12              permutateHelper( str, index + 1 );
13              swap( str[index], str[i] );
14          }
15      }
16
17      //actual function
18      public static void permutate( char[] str ){
19          permutateHelper (str, 0);
20      }
21
22      //calling the function
23      permutate("abcdefgh".toCharArray());
24
```

Case 2 : If String has duplicate

What do we do if there are duplicates in the string?

Solution 1:

The trick is to sort the characters in the alphabetical order first. We can then ignore the duplicates easily when generate the permutation.

Solution 2

?

```
1       void permutateHelperDuplicate(String prefix, String rest){
2               int N = 0;
3               if(rest!=null)
4                   N = rest.length();
5               if (N==0)
                {
```

```
            System.out.println(prefix);
        }
        else
        {
            for (int i = 0; i < rest.length(); i++)
            {


                //test if rest[i] is unique.
                boolean found = false;
                for (int j = 0; j < i; j++)
                {
                    if (rest.charAt(j) == rest.charAt(i)) //rest[j]==rest[i]
                        found = true;
                }
                if(found)
                    continue;
                String newPrefix = prefix + rest.charAt(i);
                String newRest = rest.substring(0, i) + rest.substring(i+1,N);
                permutateHelperDuplicate(newPrefix, newRest);
            }
        }
    }
```

Method 3 - Use HashSet

In this method, we can have an hashset of string, and once we generate the permutation, we add it to hashset if it is not there, otherwise we ignore it.

Please let me know if you any suggestions.

Permutation of String in Java Algorithm

To get all the permutations, we will first take out the first char from String and permute the remaining chars.
If String = “ABC”
First char = A and remaining chars permutations are BC and CB.
Now we can insert first char in the available positions in the permutations.
BC -> ABC, BAC, BCA
CB -> ACB, CAB, CBA

So we can write a recursive function call to return the permutations and then another function call to insert the first characters to get the complete list of permutations.

```
package com.journaldev.string.permutation;


import java.util.HashSet;
import java.util.Set;


/**
 * Java Program to find all permutations of a String
 * @author pankaj
 *
 */
public class StringHelper {

    public static Set<String> permutationFinder(String str) {
        Set<String> perm = new HashSet<String>();
        //Handling error scenarios
        if (str == null) {
            return null;
```

```
        } else if (str.length() == 0) {
            perm.add("");
            return perm;
        }
        char initial = str.charAt(0); // first character
        String rem = str.substring(1); // Full string without
first character
        Set<String> words = permutationFinder(rem);
        for (String strNew : words) {
            for (int i = 0;i<=strNew.length();i++){
                perm.add(charInsert(strNew, initial, i));
            }
        }
        return perm;
    }

    public static String charInsert(String str, char c, int j)
{
        String begin = str.substring(0, j);
        String end = str.substring(j);
        return begin + c + end;
    }

    public static void main(String[] args) {
        String s = "AAC";
        String s1 = "ABC";
        String s2 = "ABCD";
```

```
        System.out.println("\nPermutations for " + s + " are:
\n" + permutationFinder(s));

        System.out.println("\nPermutations for " + s1 + " are:
\n" + permutationFinder(s1));

        System.out.println("\nPermutations for " + s2 + " are:
\n" + permutationFinder(s2));

    }

}
```

Note that I have used SET to remove duplicates, so that it works for those strings also having same chars.

Here is the output of the above program:

```
Permutations for AAC are:

[AAC, ACA, CAA]


Permutations for ABC are:

[ACB, ABC, BCA, CBA, CAB, BAC]


Permutations for ABCD are:

[DABC, CADB, BCAD, DBAC, BACD, ABCD, ABDC, DCBA, ADBC, ADCB,
CBDA, CBAD, DACB, ACBD, CDBA, CDAB, DCAB, ACDB, DBCA, BDAC,
CABD, BADC, BCDA, BDCA]
```

That's all for finding all permutations of a String in java.