Given a set of characters and a positive integer k, print all possible strings of length k that can be formed from the given set.

Examples:

```
Input: 
set[] = {'a', 'b'}, k = 3

Output:
aaa
aab
aba
abb
baa
bab
bba
bbb


Input: 
set[] = {'a', 'b', 'c', 'd'}, k = 1
Output:
a
b
c
d
```

For a given set of size n, there will be n^k possible strings of length k. The idea is to start from an empty output string (we call it *prefix* in following code). One by one add all characters to *prefix*. For every character added, print all possible strings with current prefix by recursively calling for k equals to k-1.
Following is Java implementation for same.

```java
// Java program to print all possible strings of length k
class PrintAllKLengthStrings {
 
    // Driver method to test below methods
    public static void main(String[] args) {
        System.out.println("First Test");
        char set1[] = {'a', 'b'};
        int k = 3;
        printAllKLength(set1, k);
         
        System.out.println("\nSecond Test");
        char set2[] = {'a', 'b', 'c', 'd'};
        k = 1;
        printAllKLength(set2, k);
    }    
 
    // The method that prints all possible strings of length k.  It is
    //  mainly a wrapper over recursive function printAllKLengthRec()
    static void printAllKLength(char set[], int k) {
        int n = set.length;
        printAllKLengthRec(set, "", n, k);
    }
 
    // The main recursive method to print all possible strings of length k
    static void printAllKLengthRec(char set[], String prefix, int n, int k) {
         
        // Base case: k is 0, print prefix
        if (k == 0) {
            System.out.println(prefix);
            return;
        }

        // One by one add all characters from set and recursively
        // call for k equals to k-1
        for (int i = 0; i < n; ++i) {

            // Next character of input added
            String newPrefix = prefix + set[i];

            // k is decreased, because we have added a new character
            printAllKLengthRec(set, newPrefix, n, k - 1);
        }
    }
}
```

Output:

```
First Test
aaa
aab
aba
abb
baa
bab
bba
bbb

Second Test
a
b
c
d
```

The above solution is mainly generalization of this post.