CS 2401 Assignment #6

Due Date: Tuesday, October 30, 11:59PM.

(See the syllabus for late policy)

Objective: The goal of this assignment is to practice recursion.

Assignment: The assignment requires writing recursive methods for some linked list operations. **The use of loops in the recursive methods is strictly prohibited in this assignment.** That is, you cannot use for, while, and do-while in the recursive methods you will write. You can only use a loop when you will initiate values for a linked list in the main method. You will need to write a method to generate a random String. You can use loops in that method too.

Consider that you are given the following linked list node.

```
public class StringNode {
  public String head;
  public StringNode next;

  StringNode(){}

  StringNode(String s){
    head = s;
  }

  StringNode(String s, StringNode tail){
    head = s;
    next=tail;
  }

}
```

Clearly, the `StringNode` class can be used to create a linked list. Write a class named `Operations` that will contain the methods described below. In addition, some instructions are given as comments of the partial code provided later in this assignment.

1. Write a method named `getRandString(int length)` that will return a String of length given in the parameter. This method need not be a recursive one. The returned String may only contain English letters in capital (that is, letters from A to Z).
2. Write a **recursive** method named `printMyList(StringNode m)` to print all the strings in the linked list m. Notice that m is the head of the linked list.
3. Write a **recursive** method named `countKLenghthStrings (StringNode m, int k)` that will return number of Strings with length `k` in the given linked list m.
4. Write a **recursive** method named `longestStringOfMyList (StringNode m)` that will return the longest `String` in a linked list.
5. Write a **recursive** method named `lengthOfMyList (StringNode m)` that will compute and return the length of a given linked list m.
6. Write a **recursive** method named `StringNode reverseMyList (StringNode m)` to reverse a linked list. Return the head of the reversed linked list.
7. Write a **recursive** method named

`StringNode removeAStringFromMyList(StringNode m, String removee)` to remove the first occurrence of `removee` from a given linked list m.

8. Write a **recursive** method named
   `StringNode insertAStringIntoMyList(StringNode m, String insertee, int position)`
   to insert a String (insertee) into a specific position of a given linked list m. Positions start from 0 (that is, the position of the head of a list is considered 0).
9. Write a **recursive** method named
   `boolean isListInOrder(StringNode m)`
   to verify if the strings in a linked list m are lexicographically ordered. Return true if they are ordered, false otherwise.
10. Write a **recursive** method named
    `int countPalindromes(StringNode m)`
    that will count how many strings of a given linked list are palindromes. The method must call another **recursive** method named
    `boolean isPalindrome(String s)`
    to verify if a String is a palindrome. Palindrome check must NOT be case-sensitive.

A template for the **`Operations`** class is given below. You need to write your codes inside the body of the methods provided with **`Operations`**. Modify the main method for testing purpose, as necessary. The comments in the **`Operations`** class provide detailed requirements.

```
public class Operations {

  public static void main(String[] args){
    StringNode L=new StringNode("0"+getRandString(2+(int)(Math.random()*5)));
    StringNode temp = L;
    for (int i=1; i<=9;i++){
      temp.next=new StringNode(i+getRandString(2+(int)(Math.random()*5)));
      temp=temp.next;
    }

    System.out.println("All Strings in the list:");
    printMyList(L);
    System.out.println();

    boolean b = isListInOrder(L);
    System.out.println("List is ordered: "+b);
    System.out.println();

    System.out.println("Count of k-length strings");
    System.out.println("k\tNo. of Strings with length k");
    for (int k=0; k<7; k++){
      System.out.println(k+"\t"+countKLenghthStrings(L, k));
    }

    System.out.println("Longest String="+longestStringOfMyList(L));
    System.out.println("Length="+lengthOfMyList(L));

    L=reverseMyList(L);
    System.out.println("All string in the reversed list:");
    printMyList(L);
    System.out.println();


    System.out.println("Remove a given String");
    StringNode LL=removeAStringFromMyList(L, L.next.next.head);
```

```
    System.out.println("All strings in the new list:");
    printMyList(LL);
    System.out.println();

    System.out.println("All strings in the previous list:");
    printMyList(L);
    System.out.println();

    System.out.println("Insert a string in a position of the new list:");
    LL=insertAStringIntoMyList(LL, "Hello world", 3);
    printMyList(LL);
    System.out.println();

    b = isListInOrder(L);
    System.out.println("List is ordered: "+b);
    System.out.println();

    LL=insertAStringIntoMyList(LL, "ABBA", 3);
    LL=insertAStringIntoMyList(LL, "DoGeeseSeeGod", 3);

    int c = countPalindromes(LL);
    System.out.println("Found "+c+" palindromes.");

}

static String getRandString(int length) {
    /* Write your code here */
}


/* Write a recursive method to print all the strings in separate lines.
This method cannot contain any loop (that is, uses of for, while, do while
are prohibited).
*/
public static void printMyList(StringNode m){
    /* Write your code here */
}

/* Write a recursive method to retrieve the number of strings that are longer
than the length provided in the parameter. This method cannot contain any
loop (that is, uses of for, while, do while are prohibited).
*/
public static int countKLenghthStrings (StringNode m, int k){
    /* Write your code here */
}

/* Write a recursive method to retrieve the largest String from the list.
Assume that there is at least one String in the given list when the method
is called from the main function. This method cannot contain any loop (that
is, uses of for, while, do while are prohibited).
*/
public static String longestStringOfMyList (StringNode m){
    /* Write your code here */
}

/* Write a recursive method to compute the length of a list.
This method cannot contain any loop (that is, uses of for, while, do while
are prohibited).
*/
public static int lengthOfMyList (StringNode m){
    /* Write your code here */
}
```

```
  /* Write a recursive method named reverseMyList that will reverse a given
  linked list m. Return the head of the reversed linked list. It is fine
  if you need to modify the given linked list to obtain the reversed one.
  */
  public static StringNode reverseMyList (StringNode m){
    /* Write your code here */
  }

  /* Write a recursive method to remove the first occurrence of a specific
  String from a list. As an example, if your list initially contains
  AA BB CC DD BB KK and if your removee is BB, the returned list should contain
  AA CC DD BB KK after the removal. Return a new head. You must make sure that
  the parameter list m remains intact after returning the new list to the main
  method. This method cannot contain any loop (that is, uses of for, while,
  do-while are prohibited).
  */
  public static StringNode removeAStringFromMyList(StringNode m,
          String removee){
    /* Write your code here */
  }

  /* Write a recursive method to insert a String (insertee) into a specific
  position of a list. Positions start from 0 (that is, the position of
  the head of a list is 0). This method cannot contain any loop (that is,
  uses of for, while, do-while are prohibited).
  */
  public static StringNode insertAStringIntoMyList(StringNode m,
          String insertee, int position){
    /* Write your code here */

  }

  /* Write a recursive method to verify if the strings are
  lexicographically ordered in a linked list. Return true if they are
  ordered, false otherwise. This method cannot contain any loop (that is,
  uses of for, while, do-while are prohibited).
  */
  public static boolean isListInOrder(StringNode m){
    /* Write your code here */
  }

  /* Write a recursive method that will count how many strings of a given
  linked list are palindromes. The method must call another recursive
  method named isPalindrome to verify if a String is a palindrome, or
  not. Palindrome checks must NOT be case-sensitive. Neither countPalindromes
  nor isPalindrome may contain loops (that is, uses of for, while, do-while
  are prohibited).
  */
  public static int countPalindromes(StringNode m){
    /* Write your code here */
  }

  public static boolean isPalindrome(String s){
        /* Write your code here */
  }

}
```

Output of the program above is as follows. Note that a part of each string is random and hence the output may change for a different execution.

```
All Strings in the list:
0WTR
1VUX
2VZ
3SUMHD
4FIEU
5JFW
6GAKBO
7XDOF
8XTCRPF
9LH

List is ordered: true

Count of k-length strings
k        No. of Strings with length k
0        0
1        0
2        0
3        2
4        3
5        2
6        2
Longest String=8XTCRPF
Length=10
All string in the reversed list:
9LH
8XTCRPF
7XDOF
6GAKBO
5JFW
4FIEU
3SUMHD
2VZ
1VUX
0WTR

Remove a given String
All strings in the new list:
9LH
8XTCRPF
6GAKBO
5JFW
4FIEU
3SUMHD
2VZ
1VUX
0WTR

All strings in the previous list:
9LH
8XTCRPF
7XDOF
6GAKBO
5JFW
4FIEU
3SUMHD
2VZ
1VUX
0WTR

Insert a string in a position of the new list:
9LH
8XTCRPF
6GAKBO
Hello world
5JFW
4FIEU
3SUMHD
2VZ
1VUX
0WTR

List is ordered: false

Palindrome:DoGeeseSeeGod
Palindrome:ABBA
Found 2 palindromes.
```

Any use of `java.util.LinkedList` is strictly prohibited.

Deliverables: You are expected to submit two Java files (`StringNode.java` and `Operations.java`) via Blackboard. Your TA will instruct you with further details.