Merge String Alternatively

String str1 = "$\overset{0}{G}\overset{1}{E}\overset{2}{E}\overset{3}{K}$"

String str2 = "$\overset{0}{S}\overset{1}{T}\overset{2}{E}\overset{3}{R}$"

loop → 0 to 3.

String res = " "; → 0 to 3.

res += str1.charAt(0); G

res += str2.charAt(0); S res = GS

---

res += str1.charAt(1); res = GSE

res += st2.charAt(1); res = GSET

---

res += str1.charAt(2); res = GSETE

res += str2.charAt(2); res = GSETEE

---

res += str1.charAt(3) res = GSETEK

res += str2.charAt(3) res = GSETEKR.

```
S.o pln (res);
String res="  ";
for (int i=0; i<str1.length(); i++) {
    res+= str1.charAt(i);
    res+= str2.charAt(i);
  }
S.o.pln (res);
```

Long Pressed Name

String name: "Jay"

String keyboardstr: "Jaaaaaay";

Output -> **True.**

Input2.

name: "alex";

lpname: "aaleex";

Output:- True.

Solution:-

```
          0 1 2 3
name:    "a l e x"
          0 1 2 3 4 5
tpname: " a a l e e x "

int namest = 0
  Loop intpname (0 to 5)
1st Iteration  tpname.charAt(0) == name.charAt(0)
                    ↓ 'a'   namest++;
2nd Iteration  tpname. charAt(1) != name.charAt(1)
                         ↓
                         a
   ↓ 1st condition to check if char in typename
     matches with char in name
     ↓                        move to next index in
```

matches with char

↓
If matches then move to next index in both string

↓
If it doesn't match,
↳ We have 2 possibilities

① ↳ if it matches with its previous character then we can say it is fine to go ahead because it is long pressed character

② ↳ If it doesn't match with previous char then we can say string is not long pressed name
↳ return false; because we do not need to check other character

Code:-

```
String name = " alex ";        np=4
String tname = " aaleex ";

int np=0;
for(int i=0; i< tname.length(); i++) {
if(np< name.length() && tname.charAt(i)== name.charAt(np)) {
```

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ a & l & e & x \end{array} \qquad \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ a & a & l & e & e & x \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \end{array}$$

```
if(np<name.length() [illegible]
            np++;
}else if(i>0&& tname.charAt(i-1) == tname.charAt(i)){
            continue;
      }else {
         s.o.pln("false");
        return;
      }
    }
    if(np==name.length()){
     s.o.pln("true");
    }
```

Power of a String

String str = "aabbcccccdefg";

Brute force

1. Find all the substring
2. Filter the substring containing single character being repeated
3. Out of the filtered substring, get the maximum length.

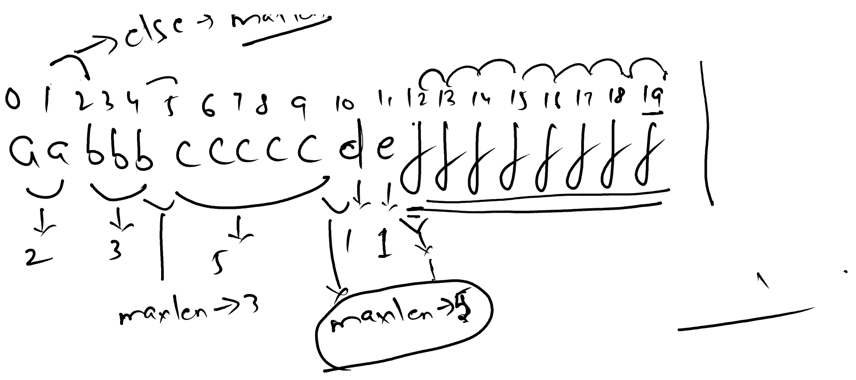
Efficient Approach

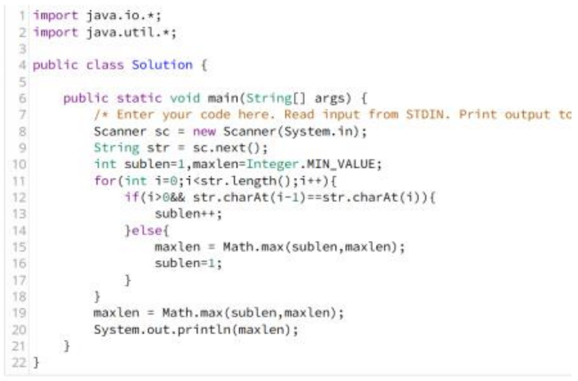
```
                 0 1 2 3 4 5 6 7 8 9 10 11 12
String  str = "a a b b c c c c c d e f g";
    int   sublen = 1, maxlen = Integer.MIN_VALUE;
for(int i=0; i<str.length(); i++) {
    if(i>0&& str.charAt(i-1) == str.charAt(i)){
            sublen++;
    }else{
     maxlen= Math.max (sublen, maxlen);
      sublen=1;
    }

  }
  maxlen = Math.max (sublen, maxlen);
  }
  S.O.Pln(maxlen);
```

→ else → maxlen → 2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 |





```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output tc
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        int sublen=1,maxlen=Integer.MIN_VALUE;
        for(int i=0;i<str.length();i++){
            if(i>0&& str.charAt(i-1)==str.charAt(i)){
                sublen++;
            }else{
                maxlen = Math.max(sublen,maxlen);
                sublen=1;
            }
        }
        maxlen = Math.max(sublen,maxlen);
        System.out.println(maxlen);
    }
}
```