Backspace Attack

Published by Matt in Java

regex scope strings

Suppose a hash **#** represents the BACKSPACE key being pressed. Write a function that transforms a string containing **#** into a string without any **#**.

Examples

```
erase("he##l#hel#llo") → "hello"

erase("major# spar##ks") → "majo spks"

erase("si###t boy") → "t boy"

erase("####") → ""
```

Notes

- In addition to characters, backspaces can also remove whitespaces.
- If the number of hashes exceeds the previous characters, remove those previous characters entirely (see example #3).
- If there only exist backspaces, return an empty string (see example #4).

LOGIC

There is no requirement to proceed in recursive manner at all, so I will resort to iterative approach.

Will perform a do while loop
Will traverse from extreme right hand side.

text.charAt(text.length()-i) //due to zero indexing, i will be initialised as 1
if it find a #, it will simply write it to the previous character.
It will then increment i.
It will then move to text.charAt(text.length() -i)
The process will terminate when it reaches text.charAt(0) (i.e. when i is equal to text.length();
It will deal with the remaining text thereafter inline with other requirements.
Since String is immutable, it will utilize a StringBuilder to ensure it can modify the content.

This appears to be nature of the challenge.
I have chosen to start on right hand side since the deletion can overwrite

existing deletions.

Test case 1: (as per the challenge)

```
String text="he##l#hel#llo";
```

```
This is final string:hello
This is original string:he##l#hel#llo
```

Test case 2: (as per the challenge)

```
String text="major# spar##ks";
```

```
This is final string:majo spks
This is original string:major# spar##ks
```

Test case 3: (as per the challenge)

```
String text="si###t boy";
```

```
This is final string:t boy
This is original string:si###t boy
```

Test case 4: (as per the challenge)

```
String text="####";
```

```
This is final string:
This is original string:####
```

Test case 5: (additional challenge)

```
String text="sit boy#";
```

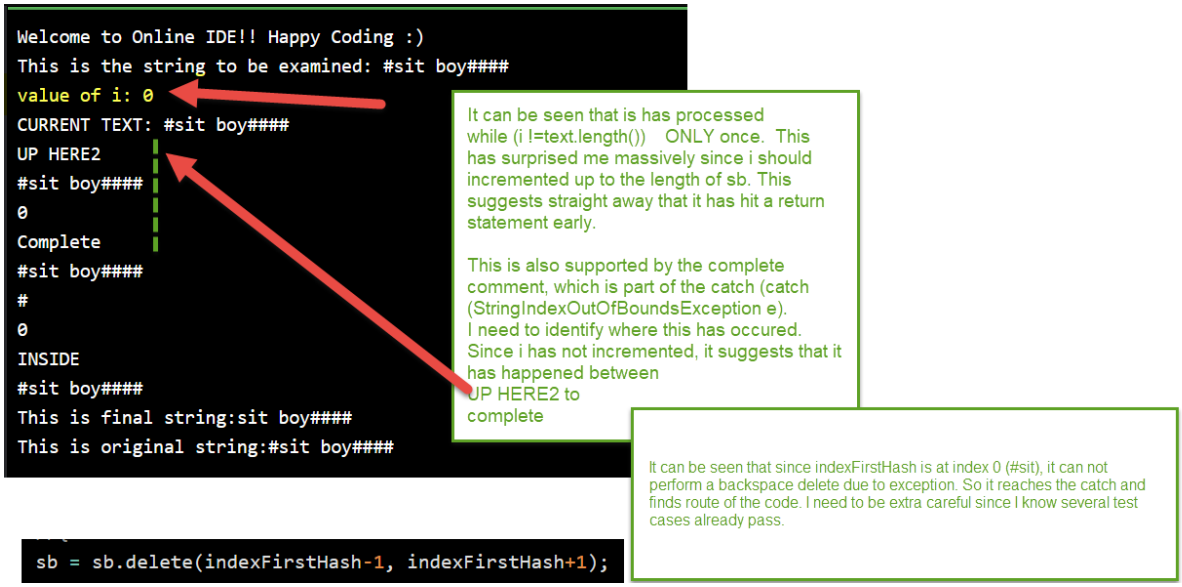
```
This is final string:sit bo
This is original string:sit boy#
```

Test case 6: (additional) - FAIL******

```
String text="#sit boy#####";
```

```
This is final string:sit boy#####
This is original string:#sit boy#####
```

I am extremely glad I have extended the testing phase.
I can clearly identify where the code would have stumbled.
I have documented it as below:



Realistically I could abandon this exercise since it has passed all the test cases provided. But I have developed sufficient skills to at least have an attempt, at minimum understand any design flaws.

This is a copy of the code which generates results above:

*********CODE – FAILING CODE ***************

```
/*
Online Java - IDE, Code Editor, Compiler

Online Java is a quick and easy tool that helps you to build, compile, test your programs online.
*/
public class Main
{
  public static void main(String[] args)
  {
    System.out.println("Welcome to Online IDE!! Happy Coding :)");


    //test cases  (from the examples)
    //String text="he##l#hel#llo";        //PASS
    //String text="major# spar##ks";    //PASS
    //String text="si###t boy";           //PASS
    //String text="####";                 //PASS
    //String text="sit boy A#mit Amlan##i"; //PASS

    //String text="#sit boy A#mit Amlan##i"; //FAIL
```

```
    //it is failing on ## in a multi-word test case  with a # in first index
    //String text="#test";  //PASS
    //String text="#test again A#mit"; //PASS
    //String text="#test again A##mit"; // FAILS - double ##
    //String text="t#est again A##mit"; // PASS
    //String text="te##sting"; //PASS
    //String text="#test a##gain A#mit"; // FAILS - double ##
    //String text="s#it boy####";  //FAIL   - giving  it #
    //String text="J#o#h#n"; //PASS


    System.out.println("This is final string:" + erase(text));
    System.out.println("This is original string:" + text);
}

public static String erase(String text)
{
    int i=0;  //iterate through StringBuilder
    StringBuilder sb = new StringBuilder(text); //places text into
    int indexLastHash;
    int indexFirstHash;

    /*
    while (sb.charAt(0)=='#')
      {
        //removes first character
        sb.delete(0,1);
      }
      */

    System.out.println("This is the string to be examined: " + sb);
    try
    {
      do
      {
        //System.out.println("****" + sb.indexOf("#"));
        //System.out.println("****" + sb.lastIndexOf("#"));

        System.out.println("value of i: " + i);
        System.out.println("CURRENT TEXT: " + sb);
```

```
  //System.out.println("This22 is last index of #:" + sb.lastIndexOf("#"));
  indexLastHash = sb.lastIndexOf("#");

  try
  {
    if (sb.charAt(indexLastHash-1)!='#')
    {
      sb = sb.delete(indexLastHash-1, indexLastHash+1);
      System.out.println("UP HERE1");

    }

    //if there is # before the existing one.
    //it will attack text from other direction.
    else
    {
      System.out.println("UP HERE2");
      System.out.println(sb);
      //since it is first hash, there can not be a hash on left hand side
      indexFirstHash = sb.indexOf("#");
      System.out.println(sb.indexOf("#"));

      //if (indexFirstHash!=0)
      //{
      sb = sb.delete(indexFirstHash-1, indexFirstHash+1);
      //}

      System.out.println("*OUTCOME***: " + sb);

    }
  }
  catch (ArrayIndexOutOfBoundsException e)
  {
    System.out.println("Reached first char");
    System.out.println(sb);
  }

  i++;

}while (i!=sb.length());
```

```
    }
    catch (StringIndexOutOfBoundsException e)
    {

      System.out.println("Complete");
      System.out.println(sb);

      //this is the scenario described in the challenge to remove any #
      //if there are excess # in relation to characters prior to it

      //This should deal with example 3 and example 4

      System.out.println(sb.charAt(0));
      System.out.println(sb.indexOf("#"));

      try
      {

       if (sb.indexOf("#")!=-1)
      {
      do
      {
        System.out.println("INSIDE");
        System.out.println(sb);
        //removes first character
        sb.delete(0,1);
      } while((sb.charAt(0)=='#'));
      }
      }
      catch (StringIndexOutOfBoundsException s)
      {
        System.out.println("sds");
        return sb.toString();
      }
    }
    return sb.toString();

  }
}
```

Just to re-iterate the issue is occurring in the following situation:
in a multi-word test case with a # in first index.
In summary, it is finding ## (as part of indexLastHash), so it then performs indexFirstHash. It removes this successfully from #sit to give sit. Then it performs following:
sb.delete(indexFirstHash-1, indexFirstHash+1);
This is effectively telling it to perform a delete operation, but clearly it should just remove the blankspace delete character itself if it finds # at indexFirstHash=0.
As a result, it is then forced into the catch statement and it gradually brings the code to closure even though i=0 (one iteration in do while loop which is trying to increase I towards sb.length().

I am inserting this logic as follows:

```
//this ensures as expected.
//it performs backspace (first argument)
//we know with following, in A#mit
//it would remove the A, but it also had to remove the #
//String text="#test a##gain A#mit"; // FAILS - double ##
//the following notation:
//sb.delete(14, 16), so it would include A# and
//exclude m

if (indexFirstHash!=0)
{
   sb.delete(indexFirstHash-1, indexFirstHash+1);
}

//in this scenario, the # would be first index such as
//#test.  It would cause exception if it attempts backspace
//delete. Instead the # is simply removed ONLY

else
{
   sb.delete(indexFirstHash, indexFirstHash+1);
```

```
}
```

I am now running through all the test scenarios again.

```
//String text="he##l#hel#llo";
 //String text="major# spar##ks";
 //String text="si###t boy";
 //String text="####";
 //String text="sit boy A#mit Amlan##i";
 //String text="#sit boy A#mit Amlan##i";
 //String text="#test";  //PASS
 //String text="#test again A#mit";
 //String text="#test again A##mit"; //
String text="t#est aga###in A##mit";
 //String text="t#est again A##mit"; //
 //String text="te##sting";
 String text="#test a###gain A#mit##";
 //String text="s#it boy####";
 //it is failing on ## in a multi-word test case
 //String text="J#o#h#n"; //PASS
```

Test case 1: (as per the challenge)

```
String text="he##l#hel#llo";
```

```
This is final string:hello
This is original string:he##l#hel#llo
```

Test case 2: (as per the challenge)

```
String text="major# spar##ks"
```

```
This is final string:majo spks
This is original string:major# spar##ks
```

Test case 3: (as per the challenge)

```
String text="si###t boy"
```

```
This is final string:t boy
This is original string:si###t boy
```

Test case 4: (as per the challenge) => **FAIL. ALTHOUGH IT PASSED PREVIOUSLY**

```
String text="####";
```

```
This is final string:##
This is original string:####
```

I will now try to manually simulate the deletion below in phases:

=> ## => I would expect it to complete erasure of all since they are ineffective backspace in absence of other characters. Also I had already included code which looked at the first index and kept on deleting it if it was a backspace. But I recall having this logic in the catch area of the code. If the code does not reach this section, there was nothing in the try to perform this action.. So it is perhaps a case of copying the code across...
I have copied the following code at the end of the mini try block (inside of the do loop):

```
if (sb.indexOf("#")!=-1)
{
  do
  {
    System.out.println("INSIDE TRY, REMOVE FIRST #");
    //removes first character
    sb.delete(0,1);
    System.out.println(sb);
  } while((sb.charAt(0)=='#'));
}
```

Test case 4a: (as per the challenge) => IT NOW PASSES

```
String text="####";
```

```
This is final string:
This is original string:####
```

Test case 5: THIS PREVIOUSLY PASSED ALSO

```
String text="sit boy A#mit Amlan##i";
```

```
This is final string:sit boy mit Amli
This is original string:sit boy A#mit Amlan##i
```

Test case 6: THIS PREVIOUSLY FAILED. ALL PASS NOW

```
String text="#sit boy A#mit Amlan##i";
```

```
This is final string:sit boy mit Amli
This is original string:#sit boy A#mit Amlan##i
```

Test case 7:

```
String text="#test";
```

```
This is final string:test
This is original string:#test
```

Test case 8:

```
String text="#test again A#mit";
```

```
This is final string:test again mit
This is original string:#test again A#mit
```

Test case 9: THIS PREVIOUSLY FAILED. ALL PASS NOW

```
String text="#test again A##mit";
```

```
This is final string:test againmit
This is original string:#test again A##mit
```

Test case 10: THIS PREVIOUSLY FAILED. ALL PASS NOW

```
String text="t#est aga###in A##mit";
```

```
This is final string:est inmit
This is original string:t#est aga###in A##mit
```

Test case 11:

```
String text="t#est again A##mit";
```

```
This is final string:est againmit
This is original string:t#est again A##mit
```

Test case 12:

```
String text="te##sting";
```

```
This is final string:sting
This is original string:te##sting
```

Test case 13: THIS PREVIOUSLY FAILED. ALL PASS NOW

```
String text="#test a###gain A#mit##"
```

```
This is final string:tesgain m
This is original string:#test a###gain A#mit##
```

Test case 14: THIS PREVIOUSLY FAILED. ALSO FAILED

```
String text="s#it boy####";
```

```
This is final string:it #
This is original string:s#it boy####
```

I will not try to manually simulate the deletion below in phases:

s#it boy#### => it boy#### => it bo### => it b## => it #

So it can be seen that it is correct... And in the logic of my coding, I am unaware of completing logic so that it can clear any excess backspaces at end or in the middle of the text.

Please note I only completed the change in Test case 4 only upon getting this failed test case. So I will test again.

Test case 14a:

```
String text="s#it boy####";
```

```
This is final string:b##
This is original string:s#it boy####
```

Unfortunately this is a complete different outcome, so I will quickly run through all my test cases again (without recording any results), to see if it has impacted them also. If it is all OK, then I can remediate this test case. Otherwise, it is a potentially huge issue.

Unfortunately it has had impact on test cases which previously passed.

For instance:

Test case 10: PREVIOUSLY PASSED

```
String text="t#est aga###in A##mit";
```

```
This is final string:est inmit
This is original string:t#est aga###in A##mit
```

now shows:

```
This is final string:nmit
This is original string:t#est aga###in A##mit
```

I will perform manual validation to be certain which is correct:

t#est aga###in A##mit => est aga###in A##mit => est ag##in A##mit => est a#in A##mit => est in A##mit => est in #mit => est inmit

So it can clearly be seen that the extra logic introduced as part of test case 4 has caused issues.

I will try to analyse it and understand at which point test case 10 entered it. This was a careless mistake, since in the catch statement I knew I was examining different circumstance in which the # resided at the front of the text,.... So I utilised:

if (sb.indexOf("#")!=-1)

In the try, this is hazardous approach in order to perform deletion at sb.charAt(0) since there is so much processing not undertaken for intermittent # backspaces.

I changed the logic to:
if (sb.indexOf("#")!==0)

It now passes again as previously:

Test case 10: THIS PREVIOUSLY FAILED. ALL PASS NOW

```
String text="t#est aga###in A##mit";
```

```
This is final string:est inmit
This is original string:t#est aga###in A##mit
```

So finally, I_have the responsibility to ensure that Test case 14 passes and that will be the end.

TEST CASE 14B: I AM HOPING TO GET EXACT SAME OUTCOME AS TEST CASE 14. I CAN THEN APPLY FURTHER LOGIC TO TRY AND CREATE THE EXPECTED OUTCOME (LOGIC WHICH HAS NOT BEEN IMPLEMENTED YET)

Fortunately it has given exact same outcome.
My logic suggests implementing anything in the try section could break the flow of the code, so I intend to complete something in the catch section (as close as possible to the return statement), in which it will just remove any extra #

```
String text="s#it boy####"
```

```
This is final string:it #
This is original string:s#it boy####
```

Infact, it might be suitable to change the existing do while loop which instructed clearance of the front #. It can perhaps clear any outstanding #

at end also.
Strictly speaking it should have no backspace effect given the locations of pending # found in test cases...

I was completely mistaken, it had to be placed in try block. I placed it right before it entered the catch.

I cut this code out of the catch:

```
if (sb.indexOf("#")!=-1)
{
  do
  {
    System.out.println("INSIDE TRY, REMOVE FIRST #");
    //removes first character
    sb.delete(0,1);
    System.out.println(sb);
  }while((sb.charAt(0)=='#'));
}
```

AND REPLACED IT WITH THE FOLLOWING AND MOVED IT INTO THE TRY:

```
if (sb.indexOf("#")!=-1)
    {
    do
    {
      System.out.println("INSIDE");
      System.out.println(sb);
      //removes first character
      sb.delete(sb.indexOf("#"),sb.indexOf("#")+1);
      count++;
    }while((count<sb.length()));
    }
```

And I finally hope all my test cases will pass. And unfortunately there are no shortcuts since I changed critical parts of the code:

Unfortunately again, there are some fail test cases, but fortunately it is a single failed case.

Test case 7:

```
String text="#test";
```

```
This is final string:test
This is original string:#test
```

There are no issues if String was ##test, since it would stay in the try loop and not cause an exception in this circumstance:
if (sb.charAt(indexLastHash-1)!='#')
It would henceforth perform the cleanup of the ineffective #

So since for String text=”#test#
it would evaluate as:
(sb.charAt(indexLastHash-1)!='#') = sb.charAt(0-1) => this would render the ArrayIndexOutOfBoundsException

Hence since it now enters the catch, I will place the code (see bottom of test case 4) back into the last catch statement.

```
if (sb.indexOf("#")!=-1)
{
   do
   {
      System.out.println("INSIDE TRY, REMOVE FIRST #");
      //removes first character
      sb.delete(0,1);
      System.out.println(sb);
   } while((sb.charAt(0)=='#'));
}
```

I will quickly run through all the test cases again. I have marked test cases on my code and will mark them as pass.
All are now successful, here are some outputs of test cases.

TEST CASE 1:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: he##l#hel#llo

1Deleting character: l at index: 8

Formatted text: he##l#hello

1Deleting character: l at index: 4

Formatted text: he##hello

2Deleting character: e at index: 1

Formatted text: h#hello

1Deleting character: h at index: 0

Formatted text: hello

*** This is final string:<hello>

*** This is original string:<he##l#hel#llo>

TEST CASE 2:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: major# spar##ks

2Deleting character: r at index: 4

Formatted text: majo spar##ks

2Deleting character: r at index: 8

Formatted text: majo spa#ks

1Deleting character: a at index: 7

Formatted text: majo spks

*** This is final string:<majo spks>

*** This is original string:<major# spar##ks>

TEST CASE 3:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: si###t boy

2Deleting character: i at index: 1

Formatted text: s##t boy

2Deleting character: s at index: 0

Formatted text: #t boy

2Removing ineffective BACKSPACE at index 0

Formatted text: t boy

*** This is final string:<t boy>

*** This is original string:<si###t boy>

TEST CASE 4:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: ####

1Removing ineffective BACKSPACE at index: 0

Formatted text: ###

2Removing ineffective BACKSPACE at index 0

Formatted text: ##

2Removing ineffective BACKSPACE at index 0

Formatted text: #

2Removing ineffective BACKSPACE at index 0

Formatted text:

*** This is final string:<>

*** This is original string:<####>

TEST CASE 5:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: sit boy A#mit Amlan##i

2Deleting character: A at index: 8

Formatted text: sit boy mit Amlan##i

2Deleting character: n at index: 16

Formatted text: sit boy mit Amla#i

1Deleting character: a at index: 15

Formatted text: sit boy mit Amli

*** This is final string:<sit boy mit Amli>

*** This is original string:<sit boy A#mit Amlan##i>

TEST CASE 6:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: #sit boy A#mit Amlan##i

1Removing ineffective BACKSPACE at index: 0

Formatted text: sit boy A#mit Amlan##i

2Deleting character: A at index: 8

Formatted text: sit boy mit Amlan##i

2Deleting character: n at index: 16

Formatted text: sit boy mit Amla#i

1Deleting character: a at index: 15

Formatted text: sit boy mit Amli

*** This is final string:<sit boy mit Amli>

*** This is original string:<#sit boy A#mit Amlan##i>

TEST CASE 7:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: #test

5Removing ineffective BACKSPACE at index 0

Formatted text: test

*** This is final string:<test>

*** This is original string:<#test>

TEST CASE 8:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: #test again A#mit

1Deleting character: A at index: 12

Formatted text: #test again mit

2Removing ineffective BACKSPACE at index 0

Formatted text: test again mit

*** This is final string:<test again mit>

*** This is original string:<#test again A#mit>

TEST CASE 9:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: #test again A##mit

1Removing ineffective BACKSPACE at index: 0

Formatted text: test again A##mit

2Deleting character: A at index: 11

Formatted text: test again #mit

1Deleting character: at index: 10

Formatted text: test againmit

*** This is final string:<test againmit>

*** This is original string:<#test again A##mit>

TEST CASE 10:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: t#est aga###in A##mit

2Deleting character: t at index: 0

Formatted text: est aga###in A##mit

2Deleting character: a at index: 6

Formatted text: est ag##in A##mit

2Deleting character: g at index: 5

Formatted text: est a#in A##mit

2Deleting character: a at index: 4

Formatted text: est in A##mit

2Deleting character: A at index: 7

Formatted text: est in #mit

1Deleting character: at index: 6

Formatted text: est inmit

*** This is final string:<est inmit>

*** This is original string:<t#est aga###in A##mit>

TEST CASE 11:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: t#est again A##mit

2Deleting character: t at index: 0

Formatted text: est again A##mit

2Deleting character: A at index: 10

Formatted text: est again #mit

1Deleting character: at index: 9

Formatted text: est againmit

*** This is final string:<est againmit>

*** This is original string:<t#est again A##mit>

TEST CASE 12:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: te##sting

2Deleting character: e at index: 1

Formatted text: t#sting

1Deleting character: t at index: 0

Formatted text: sting

*** This is final string:<sting>

*** This is original string:<te##sting>

TEST CASE 13:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: #test a###gain A#mit##

1Removing ineffective BACKSPACE at index: 0

Formatted text: test a###gain A#mit##

2Deleting character: a at index: 5

Formatted text: test ##gain A#mit##

2Deleting character: at index: 4

Formatted text: test#gain A#mit##

2Deleting character: t at index: 3

Formatted text: tesgain A#mit##

2Deleting character: A at index: 8

Formatted text: tesgain mit##

2Deleting character: t at index: 10

Formatted text: tesgain mi#

1Deleting character: i at index: 9

Formatted text: tesgain m

*** This is final string:<tesgain m>

*** This is original string:<#test a###gain A#mit##>

TEST CASE 14:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: s#it boy####

2Deleting character: s at index: 0

Formatted text: it boy####

2Deleting character: y at index: 5

Formatted text: it bo###

2Deleting character: o at index: 4

Formatted text: it b##

2Deleting character: b at index: 3

Formatted text: it #

3Deleting character: at index: 2

Formatted text: it

*** This is final string:<it>

*** This is original string:<s#it boy####>

FINAL TEST CASE:

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##c#o#d#e#.

1Deleting character: e at index: 67

Formatted text: This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##c#o#d#.

1Deleting character: d at index: 65

Formatted text: This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##c#o#.

1Deleting character: o at index: 63

Formatted text: This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##c#.

1Deleting character: c at index: 61

Formatted text: This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: at index: 4

Formatted text: Thisi#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: i at index: 4

Formatted text: Thiss# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: s at index: 4

Formatted text: This a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: v at index: 7

Formatted text: This a ##r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: at index: 6

Formatted text: This a#r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: a at index: 5

Formatted text: This r#y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: r at index: 5

Formatted text: This y lo#n##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: o at index: 8

Formatted text: This y ln##g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: n at index: 8

Formatted text: This y l#g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: l at index: 7

Formatted text: This y g###sentence t#o# s#e#e# t#h### ##.

2Deleting character: g at index: 7

Formatted text: This y ##sentence t#o# s#e#e# t#h### ##.

2Deleting character: at index: 6

Formatted text: This y#sentence t#o# s#e#e# t#h### ##.

2Deleting character: y at index: 5

Formatted text: This sentence t#o# s#e#e# t#h### ##.

2Deleting character: t at index: 14

Formatted text: This sentence o# s#e#e# t#h### ##.

2Deleting character: o at index: 14

Formatted text: This sentence s#e#e# t#h### ##.

2Deleting character: s at index: 15

Formatted text: This sentence e#e# t#h### ##.

2Deleting character: e at index: 15

Formatted text: This sentence e# t#h### ##.

2Deleting character: e at index: 15

Formatted text: This sentence t#h### ##.

2Deleting character: t at index: 16

Formatted text: This sentence h### ##.

2Deleting character: h at index: 16

Formatted text: This sentence ## ##.

2Deleting character: at index: 15

Formatted text: This sentence # ##.

2Deleting character: at index: 14

Formatted text: This sentence ##.

2Deleting character: at index: 14

Formatted text: This sentence #.

1Deleting character: at index: 13

Formatted text: This sentence.

*** This is final string:<This sentence.>

*** This is original string:<This #i#s# a v###r#y lo#n##g###sentence t#o# s#e#e# t#h### ##c#o#d#e#.>

FINAL TEST CASE:

```
String text="J#o#h#n";
```

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: J#o#h#n

1Deleting character: h at index: 4

Formatted text: J#o#n

1Deleting character: o at index: 2

Formatted text: J#n

1Deleting character: J at index: 0

Formatted text: n

*** This is final string:<n>

*** This is original string:<J#o#h#n>

FINAL TEST CASE:

```
String text="   ###"
```

Welcome to Online IDE!! Happy Coding :)

This is the string to be examined: ###

2Deleting character: at index: 2

Formatted text: ##

2Deleting character: at index: 1

Formatted text: #

3Deleting character: at index: 0

Formatted text:

*** This is final string:<>

*** This is original string:< ###>