********** OUTPUT **********

```
Welcome to Online IDE!! Happy Coding :)
Approximate number strings: 4
should reach here
Current segment value: null
Exception in thread "main"
java.lang.NullPointerException
    at stringExercise.checkString(Main.java:93)
    at stringExercise.<init>(Main.java:32)
```

```
    at Main.main(Main.java:16)


** Process exited - Return Code: 1 **
```

The lines affected are highlighted below:

*** 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 :)");

      String sample = "the quick brown fox jumps over the laxy dog";

      int lengthLine = 10;

      stringExercise  se = new stringExercise(sample, lengthLine);

        }
}


class stringExercise
{
   private String sample;
   private int lengthLine;

   public stringExercise(String sample, int lengthLine)
```

```
   {
      this.sample=sample;
      this.lengthLine=lengthLine;

      System.out.println(checkString());

   }


   public String checkString()
   {
      //no way to break the text;
      if ((sample.length()>lengthLine) &&  (sample.indexOf(" ")==-1))
         {
            System.out.println("contigious text longer than limit: " + lengthLine);
            return null;

         }

      //if the string is less than or equal segment specified, it will return entire string
      if (sample.length()<=lengthLine)
      {
         System.out.println("contigious text shorter or equal to limit: " + lengthLine);
         return sample;

      }



      // if the segment is greater or equal to segment and there is a space in text....
      // most processing will occur here




      /*
      ********* NOTE NO CONTENT HAS BEEN WRITTEN TO ANY STRINGS UP TO THIS
POINT!!!!! ***************
      */


      double stringsRequired;

      stringsRequired=Math.ceil(sample.length()/lengthLine);  // to get approximation number
strings
      int stringsRequiredConverted = (int) stringsRequired;
      System.out.println("Approximate number strings: " + stringsRequiredConverted);

      if ((sample.length()>lengthLine) &&  (sample.indexOf(" ")!=-1))
         {
            System.out.println("should reach here");

            String output[] =  new String[stringsRequiredConverted];
```

```
int init=0;
int storePos=0;
int charsToRewrite=0;

char[] charArray;

int segment = 0;

   for (int k=0; k<sample.length();k++)  //processing each char in sample
   {
      System.out.println("Current segment value: " + output[segment]);


      if (output[segment].isEmpty())  //if string is empty, then need to ensure null not
appended
      { // an example might be if it is processing the first character in the segment

         output[segment] = Character.toString(sample.charAt(k));
      }


      else
      {
         output[segment] = output[segment] + Character.toString(sample.charAt(k));
      }


      System.out.println(output[segment]);


       if (sample.charAt(k)==' ')
         {
            storePos=k;  // this keeps a track of the previous blank space
            // this is in event that a word is truncated and spans two lines
            // and allows a roll back
         }

      //block segments are lengthLine;
      if ((lengthLine % (k+1))== 0)   //this tests if k has reached the last position in the
segment
      // 1 is added to k since it is zero index
      //there should be no remainder

      System.out.println("This is current segment: " + segment);

      {
         if (sample.charAt(k)==' ')
         {
            //this is acceptable since it means that a new word will start in new segment
            // since this is exactly 1 space between words
            System.out.println(output[segment]);
            segment++;
            continue;
```

```
                }

                if (sample.charAt(k)!=' '  &&  sample.charAt(k+1)==' ' )
                {
                  //this is acceptable since it means that segment will end with char and new
segment start with space
                  System.out.println(output[segment]);
                  segment++;
                  continue;
                }


                // this is slightly trickier, since the word should not be truncated


                if (sample.charAt(k)!=' '  &&  sample.charAt(k+1)!=' ' )
                {
                  //new string will have to start one position after storePos
                  // and all characters written in current segment from ' ' onwards will need to
be wiped
                  // and k the counter will also need to be rolled backwards!


                  //converted the existing segment into character array since can not replace
characters
                  // in a string
                  charArray = output[segment].toCharArray();


                  for (int m=storePos+1; m<lengthLine;m++)
                  {


                    charArray[m]=' ';

                    // now need to convert the modified character array back into String
                    output[segment] = String.valueOf(charArray);

                    System.out.println(output[segment]);
                    segment++;

                    // k the counter can be reset as such
                    //k=k-charsToRewrite;

                    //preferred route of resetting k is as follows
                    k=storePos+1;
                    continue;
                  }

                }

              }
            }
        }
```

```
        return null;

        }


}
```