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

TEST SCENARIOS (USING NO CLASS LEVEL VARIABLES)

```
expandedForm(5.44) → "5 + 4/10 + 4/100"
```

```
Welcome to Online IDE!! Happy Coding :)
IN HERE**********

5 + 4/100 + 4/10


** Process exited - Return Code: 0 **
```

```
expandedForm(123.025) → "100 + 20 + 3 + 2/100 + 5/1000"
```

```
Welcome to Online IDE!! Happy Coding :)
IN HERE**********
IN HERE**********
IN HERE**********

100 + 20 + 3 + 5/1000 + 2/100 + 0/10
```

```
25.24 → "20 + 5 + 2/10 + 4/100"
```

```
Welcome to Online IDE!! Happy Coding :
IN HERE**********
IN HERE**********

20 + 5 + 4/100 + 2/10
```

70701.05 → "70000 + 700 + 1 + 5/100"

```
Welcome to Online IDE!! Happy Coding :)
IN HERE*************
IN HERE*************
IN HERE*************

70000 + 700 + 1 + 5/100 + 0/10
```

685.27 → "600 + 80 + 5 + 2/10 + 7/100"

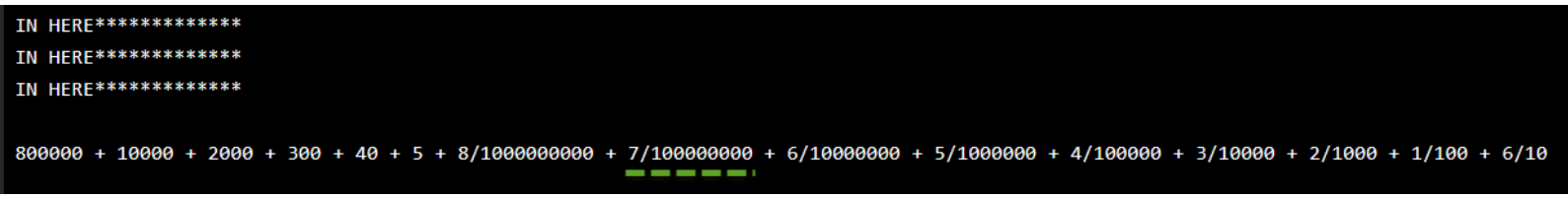
```
Welcome to Online IDE!! Happy Coding :)
IN HERE*************
IN HERE*************
IN HERE*************

600 + 80 + 5 + 7/100 + 2/10


** Process exited - Return Code: 0 **
```

An extreme test scenario:
81234594.612345678

```
IN HERE*************
IN HERE*************
IN HERE*************
IN HERE*************

8 + 1000000 + 200000 + 30000 + 4000 + 500 + 90 + 4 + 8/100000000 + 6/10000000 + 5/1000000 + 4/100000 + 3/10000 + 2/1000 + 1/100 + 6/10
```

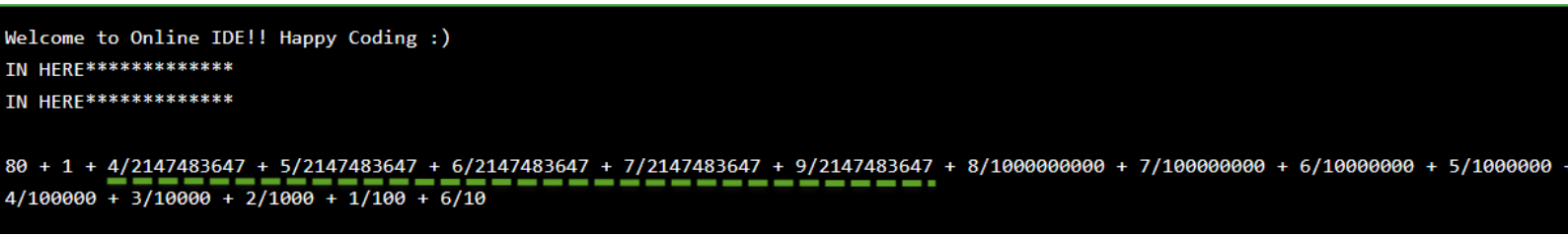
Was expecting numerator of 7

Problem fixed by shortening the whole number part:
812345.612345678



It will also reach the limit of denominator if too much precision is required:

81.6123456789765432



/
*** 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.
*/

// can not use full recursion...

public class Main
```

```
{
   public static void main(String[] args)
   {
      System.out.println("Welcome to Online IDE!! Happy Coding :)");

      System.out.println("\n" + expandedForm(81.6123456789765432));  //input value
   }

   public static String expandedForm(Double number)
   {
      String createString=null;  // this is used to hold part return value

      String convertedNumber = Double.toString(number);  // number requires conversion to
a String

      //problem has to be treated as two parts... string before and after decimal place.
      // solved recursively as oppose to iteratively.
      // it has impacted certain areas notably the fractional part..

      int positionDecimalPoint= convertedNumber.indexOf('.');   //zero based notation
      int remainingChars;
      int roundedNearestUnit;  // for instance  652 becomes  600 + 50  + 2
      int count=1;  ///this will not increment

      Double remainingPart=0.0;  //used extensively to hold the remaining portion of initial
number

      String charsBeforeDecimal = convertedNumber.substring(0,positionDecimalPoint);
      String charsAfterDecimal =
convertedNumber.substring(positionDecimalPoint+1,convertedNumber.length());

      int lengthCharsBeforeDecimal = charsBeforeDecimal.length();  // this will be used to
permit 10 power.
      // for instance  6 x pow (10, (3-1)  = 600
      int lengthCharsAfterDecimal =  charsAfterDecimal.length();  // this will be used fractional
part

      //System.out.println(charsBeforeDecimal);
      //System.out.println(charsAfterDecimal);


         int prefix;     // this is used to check the first digit in remaining number of if a decimal
         String format=null;  //
         int denominator;  //denominator
         char numerator;   //numerator
         String finalNumber;
         String temp;   // this will hold value of value before substring method is applied


         // when trying to cast, unfortunately it failed to pick up first Character
         // it picked up two characters....
         //int prefix = (int)(convertedNumber.charAt(0));
```

```
      // do while loop will execute until length number is greater than 1
      do
      {
         //System.out.println("conversion: " + convertedNumber);
         remainingChars = convertedNumber.length();




         //System.out.println("prefix22:  " + prefix);
         //System.out.println("******* " + remainingPart);
         //System.out.println("LENGTH***: " + convertedNumber.length());
         //System.out.println("LENGTH after decimal***: " + lengthCharsAfterDecimal);

         temp=convertedNumber;

         //System.out.println("$$$$$$$$$$$$: " + convertedNumber);

         prefix = Character.getNumericValue(temp.charAt(0));

         convertedNumber=convertedNumber.substring(1,remainingChars);  // it will truncate
first char.
         //remainingChars is the length, since its zero based index, this becomes correct
exclusion

         remainingPart = Double.parseDouble(convertedNumber);

         //System.out.println("This i ccurrent length:" + convertedNumber.length());


      //System.out.println("CHARS AFTER DECIMAL:" + lengthCharsAfterDecimal);


         if (prefix==-1)   // since its not a char that can be converted to a number... a decimal
point
         {
            if (lengthCharsAfterDecimal==1)  // this brings closure since final digit after decimal
point
            {
               //System.out.println("CLOSURE");

               denominator = 10;  // the first digit after decimal point represents tenth

               finalNumber = temp.charAt(1) + "/" + denominator;   // temp is being used since
convertednumber has already
               //undergone a truncation outside of this loop....

               //System.out.println(convertedNumber);

               //System.out.println("This is final part: " + temp);
```

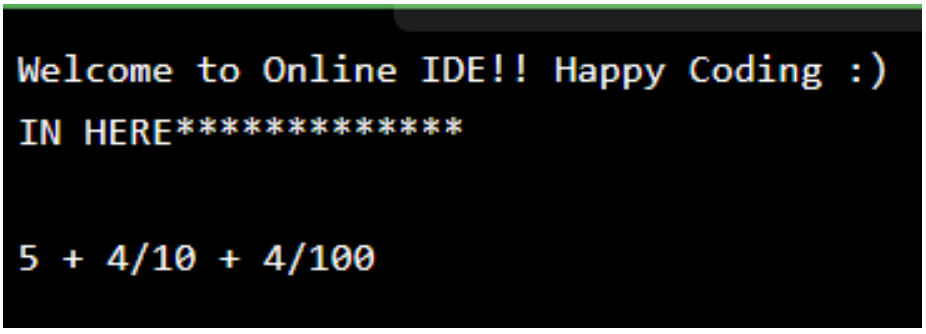
```
            //return createString + " + " + finalNumber;
            return finalNumber;
        }
        // need to see how far away from decimal point
        //index is 0 of decimal point...

        denominator = (int) Math.pow(10,lengthCharsAfterDecimal);    //
lengthCharsAfterDecimal will get smaller since truncation occuring on
                                                //right hand outside

        numerator = convertedNumber.charAt(convertedNumber.length()-1);
        //numerator is based on the last character

        format =  numerator + "/" + denominator;
        //System.out.println("after decimal:" + format);
        //System.out.println("REEEMAIJING: " + remainingChars);

        convertedNumber=temp.substring(0,temp.length()-1);  //the new remaining
number is truncated by single digit at end

        //System.out.println("NEW NEW NEW: " + convertedNumber);

        remainingPart = Double.parseDouble(convertedNumber);  //number is cast to a
double to allow recursion

        //System.out.println("NEW NEW NEW REMAINING: " + remainingPart);
        //System.out.println("THIS IS CREATE STRING: " + createString);
        //System.out.println("LEFT: " + remainingPart);



        return format + " + " + expandedForm(remainingPart);  //recursive call
    }

    //System.out.println("CHECK********************");
    //System.out.println("length number: " + convertedNumber.length());
    //System.out.println("length decimal: " +lengthCharsAfterDecimal);
    //System.out.println("length decimal before: " +lengthCharsBeforeDecimal);

    //System.out.println("HAT IS HERE" + temp.charAt(0));

    // This ensures there is digit before decimal to process
    //Also it ensures that the digit before is not a 0... since it would have already
processed it....
    // this is more important since when it reaches  .4  for instance, since the number is
declared
    // a double, Java adds 0 at front into  0.4.... so it needs to ensures it does not reach
here if this
    // is the case....
    if (lengthCharsBeforeDecimal>=1 && temp.charAt(0)!='0')

    {
```

```
            System.out.println("IN HERE**************");

        // Now remaining characters is used in  10 ^  remainingChars
        // for instance  247  will give  2 x (10 ^ 2) = 200
        // count is always one to exclude the number being processed

        roundedNearestUnit =  (int) (Math.pow(10,lengthCharsBeforeDecimal-count) * prefix);
        //System.out.println("rounded down:" + roundedNearestUnit);
        //System.out.println("processing: " + convertedNumber);

        createString = Integer.toString(roundedNearestUnit);  // Integer converted to a string

        //System.out.println("This is remaining part:" + remainingPart);

        String remainder = remainingPart.toString();


        // This first condition implies it has not reached the decimal portion since
        // first char is not the decimal point.
        // it uses string created from above... createString
        // it will then continue recursive loop to ascertain if conditions are met again....

        // the next part is if the condition evaluates to false.... (i.e it has reached decimal point
section)
        // it would use createstring as well as perform recursive call again....

        return createString + " + " +  expandedForm(remainingPart);

      }

   }while(convertedNumber.length()>1);

      return " ";   // it should not really reach here....

   }

}
```

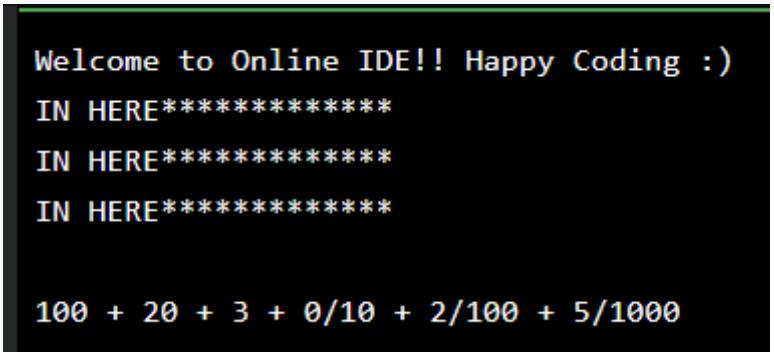
TEST SCENARIOS (USING 1 CLASS LEVEL VARIABLE), SO USING INCREMENTED VARIABLE..
BREAKS THE CYCLE OF FULL RECURSION.

NOW ALLOWING SIGNIFICANT BITS AFTER DECIMAL IN EXPECTED ORDER
THE CODE HAS BEEN MODIFIED:
SAME TESTS CONDUCTED:

```
expandedForm(5.44) → "5 + 4/10 + 4/100"
```




```
expandedForm(123.025) → "100 + 20 + 3 + 2/100 + 5/1000"
```




```
25.24 → "20 + 5 + 2/10 + 4/100"
```

```
Welcome to Online IDE!! Happy Coding :)
IN HERE*************
IN HERE*************

20 + 5 + 2/10 + 4/100
```

70701.05 → "70000 + 700 + 1 + 5/100"

```
Welcome to Online IDE!! Happy Coding :)
IN HERE*************
IN HERE*************
IN HERE*************

70000 + 700 + 1 + 0/10 + 5/100
```

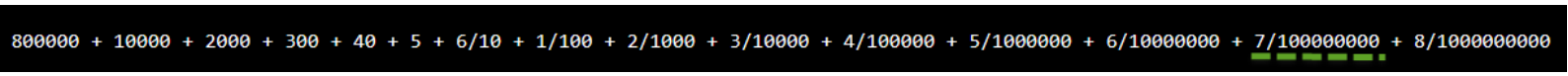
685.27 → "600 + 80 + 5 + 2/10 + 7/100"

```
Welcome to Online IDE!! Happy Coding :)
IN HERE*************
IN HERE*************
IN HERE*************

600 + 80 + 5 + 2/10 + 7/100
```

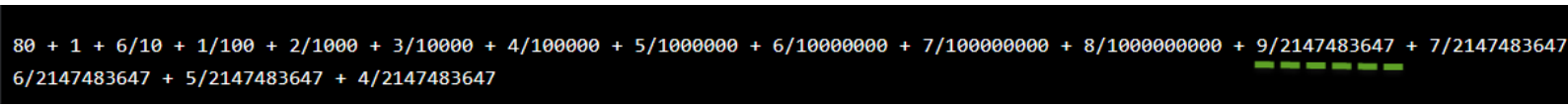
An extreme test scenario:
81234594.612345678

```
8 + 1000000 + 200000 + 30000 + 4000 + 500 + 90 + 4 + 6/10 + 1/100 + 2/1000 + 3/10000 + 4/100000 + 5/1000000 + 6/10000000 + 8/100000000

** Process exited - Return Code: 0 **
```

Similar to pure recursive (without any class variables, was expecting numerator of 7



Problem fixed by shortening the whole number part:
812345.612345678



It will also reach the limit of denominator if too much precision is required:

81.6123456789765432

/
*** 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.
*/

// can not use full recursion...

public class Main
{
   static int posAfterDecimal=1;   // class level variable to support the portion after decimal

   public static void main(String[] args)
   {
      System.out.println("Welcome to Online IDE!! Happy Coding :)");

      System.out.println("\n" + expandedForm(81.6123456789765432));  //input value
```

```
   }

   public static String expandedForm(Double number)
   {
      String createString=null;  // this is used to hold part return value

      String convertedNumber = Double.toString(number);  // number requires conversion to
a String

      //problem has to be treated as two parts... string before and after decimal place.
      // solved recursively as oppose to iteratively.
      // it has impacted certain areas notably the fractional part..

      int positionDecimalPoint= convertedNumber.indexOf('.');   //zero based notation
      int remainingChars;
      int roundedNearestUnit;  // for instance  652 becomes  600 + 50  + 2
      int count=1;  ///this will not increment

      Double remainingPart=0.0;  //used extensively to hold the remaining portion of initial
number

      String charsBeforeDecimal = convertedNumber.substring(0,positionDecimalPoint);
      String charsAfterDecimal =
convertedNumber.substring(positionDecimalPoint+1,convertedNumber.length());

      int lengthCharsBeforeDecimal = charsBeforeDecimal.length();  // this will be used to
permit 10 power.
      // for instance  6 x pow (10, (3-1)  = 600
      int lengthCharsAfterDecimal =  charsAfterDecimal.length();  // this will be used fractional
part

      //System.out.println(charsBeforeDecimal);
      //System.out.println(charsAfterDecimal);

int prefix;     // this is used to check the first digit in remaining number of if a decimal
         String format=null;  //
         int denominator;  //denominator
         char numerator;   //numerator
         String finalNumber;
         String temp;   // this will hold value of value before substring method is applied


         // when trying to cast, unfortunately it failed to pick up first Character
         // it picked up two characters....
         //int prefix = (int)(convertedNumber.charAt(0));




      // do while loop will execute until length number is greater than 1
      do
      {
         //System.out.println("conversion: " + convertedNumber);
```

```
        remainingChars = convertedNumber.length();




        //System.out.println("prefix22:  " + prefix);
        //System.out.println("******* " + remainingPart);
        //System.out.println("LENGTH***: " + convertedNumber.length());
        //System.out.println("LENGTH after decimal***: " + lengthCharsAfterDecimal);

        temp=convertedNumber;

        //System.out.println("$$$$$$$$$$$$: " + convertedNumber);

        prefix = Character.getNumericValue(temp.charAt(0));

        convertedNumber=convertedNumber.substring(1,remainingChars);  // it will truncate
first char.
        //remainingChars is the length, since its zero based index, this becomes correct
exclusion

        remainingPart = Double.parseDouble(convertedNumber);

        //System.out.println("This i ccurrent length:" + convertedNumber.length());


      //System.out.println("CHARS AFTER DECIMAL:" + lengthCharsAfterDecimal);


        if (prefix==-1)   // since its not a char that can be converted to a number... a decimal
point
        {

           //System.out.println("power: " + posAfterDecimal);

           if (lengthCharsAfterDecimal==1)  // this brings closure since final digit after decimal
point
           {
              //System.out.println("CLOSURE");

              denominator = (int)Math.pow(10,posAfterDecimal);  // the first digit after decimal
point represents tenth

              finalNumber = temp.charAt(1) + "/" + denominator;   // temp is being used since
convertednumber has already
              //undergone a truncation outside of this loop....

              //System.out.println(convertedNumber);

              //System.out.println("This is final part: " + temp);
```

```
            createString="";

            return finalNumber;
        }
        // need to see how far away from decimal point
        //index is 0 of decimal point...

        //posAfterDecimal++;

        denominator = (int) Math.pow(10,posAfterDecimal);    // it is initially set to 1.. so pow(10,1) = tenth

        numerator = temp.charAt(1);   //decimal point is first position


        format =  numerator + "/" + denominator;
        //System.out.println("numerator is: " + numerator);
        //System.out.println("denominator is: " + denominator);

        //System.out.println("after decimal:" + format);
        //System.out.println("REEEMAIJING: " + remainingChars);

        //System.out.println("This is temp: " + temp);


        // This is slightly more involved since it needs start from 2nd digit after decimal point.
        // It will remove decimal point and this needs to be appended manually

        convertedNumber="." + temp.substring(2,temp.length());  //the new remaining number is truncated by single digit at end

        //System.out.println("NEW NEW NEW: " + convertedNumber);


        remainingPart = Double.parseDouble(convertedNumber);  //number is cast to a double to allow recursion

        posAfterDecimal++;  // this is to ensure digit can be raised to correct power

        //System.out.println("NEW NEW NEW REMAINING: " + remainingPart);
        //System.out.println("THIS IS CREATE STRING: " + createString);
        //System.out.println("LEFT: " + remainingPart);

        return format + " + " + expandedForm(remainingPart);  //recursive call
    }

    //System.out.println("CHECK********************");
    //System.out.println("length number: " + convertedNumber.length());
    //System.out.println("length decimal: " +lengthCharsAfterDecimal);
    //System.out.println("length decimal before: " +lengthCharsBeforeDecimal);

    //System.out.println("HAT IS HERE" + temp.charAt(0));
```

```
        // This ensures there is digit before decimal to process
        //Also it ensures that the digit before is not a 0... since it would have already
processed it....
        // this is more important since when it reaches  .4  for instance, since the number is
declared
        // a double, Java adds 0 at front into  0.4.... so it needs to ensures it does not reach
here if this
        // is the case....
        if (lengthCharsBeforeDecimal>=1 && temp.charAt(0)!='0')

        {
           System.out.println("IN HERE**************");

        // Now remaining characters is used in  10 ^  remainingChars
        // for instance  247  will give  2 x (10 ^ 2) = 200
        // count is always one to exclude the number being processed

        roundedNearestUnit =  (int) (Math.pow(10,lengthCharsBeforeDecimal-count) * prefix);
        //System.out.println("rounded down:" + roundedNearestUnit);
        //System.out.println("processing: " + convertedNumber);

        createString = Integer.toString(roundedNearestUnit);  // Integer converted to a string

        //System.out.println("This is remaining part:" + remainingPart);

        String remainder = remainingPart.toString();


        // This first condition implies it has not reached the decimal portion since
        // first char is not the decimal point.
        // it uses string created from above... createString
        // it will then continue recursive loop to ascertain if conditions are met again....

        // the next part is if the condition evaluates to false.... (i.e it has reached decimal point
section)
        // it would use createstring as well as perform recursive call again....

        return createString + " + " + expandedForm(remainingPart);

      }

   }while(convertedNumber.length()>1);

      return " ";   // it should not really reach here....

   }

}
```