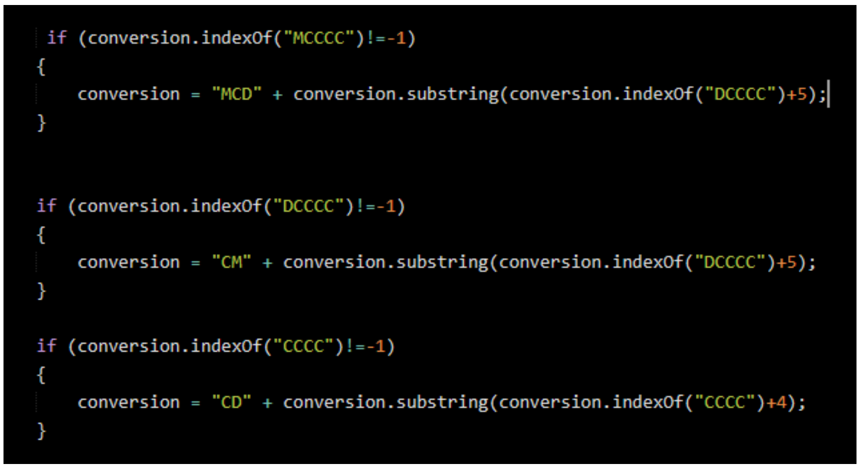
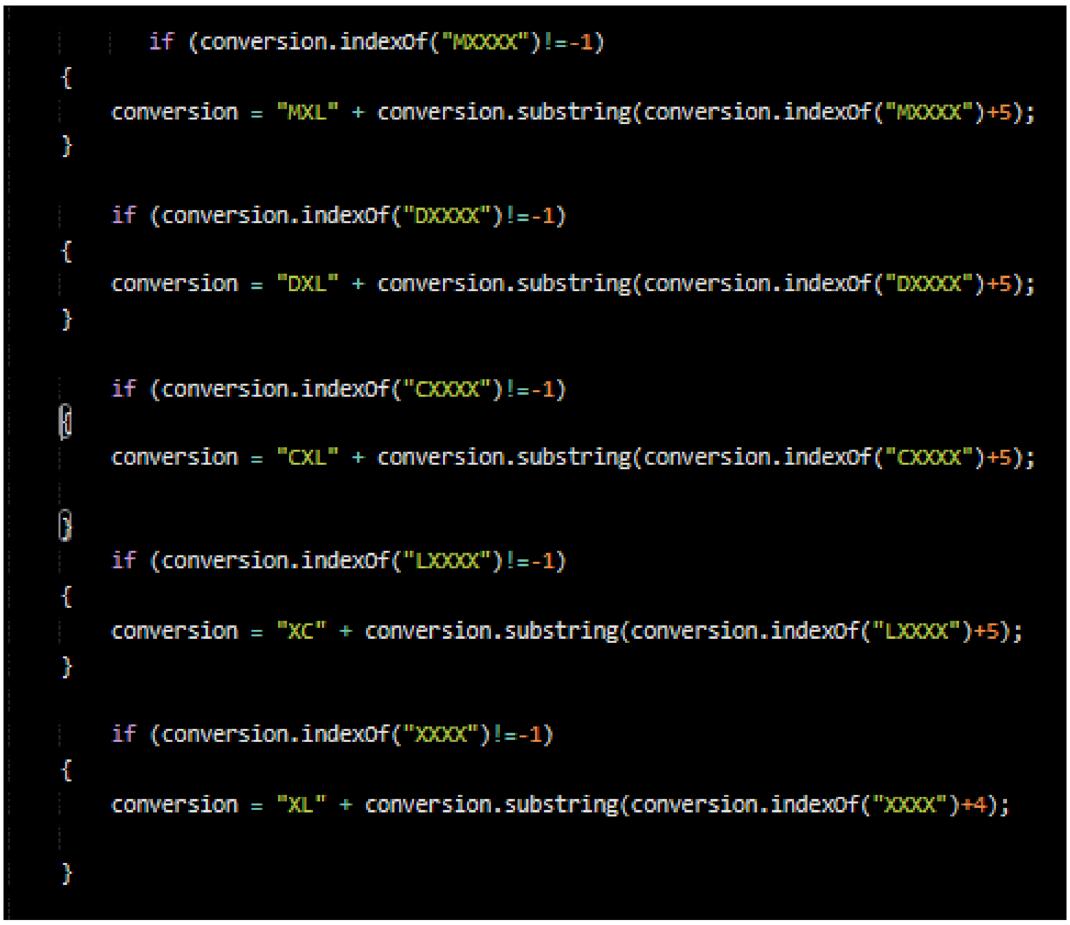
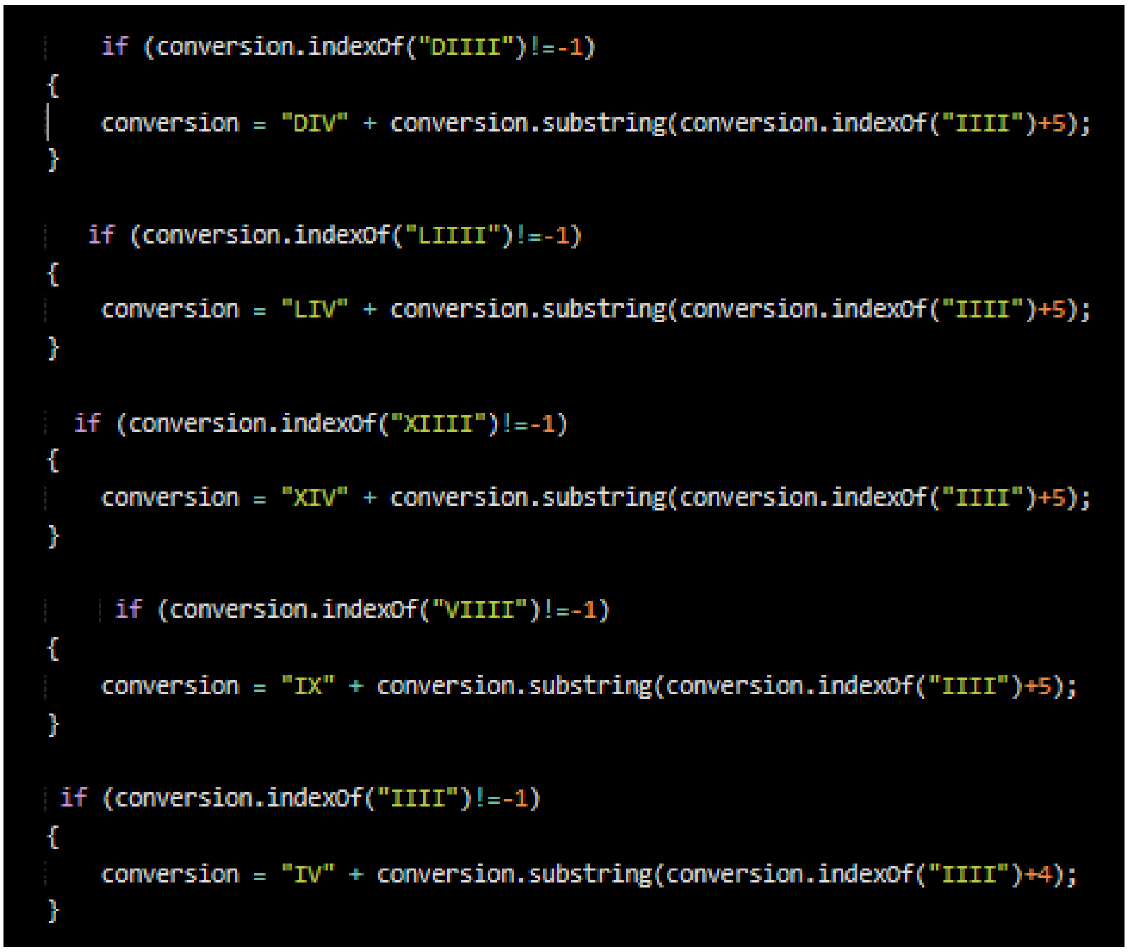
I have now started again and placed in all my implementations as follows.
To bring this to a close as quick as possible, I am just going to execute all conversions from 1=>4000 initially. Note 4000 is limit due to change in convention henceforth..



```
if (conversion.indexOf("MCCCC")!=-1)
{
    conversion = "MCD" + conversion.substring(conversion.indexOf("DCCCC")+5);
}


if (conversion.indexOf("DCCCC")!=-1)
{
    conversion = "CM" + conversion.substring(conversion.indexOf("DCCCC")+5);
}

if (conversion.indexOf("CCCC")!=-1)
{
    conversion = "CD" + conversion.substring(conversion.indexOf("CCCC")+4);
}
```

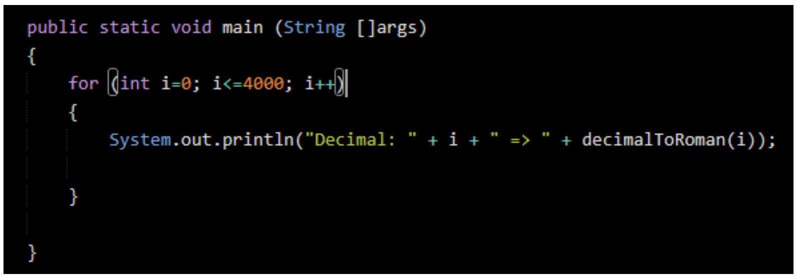


```
    if (conversion.indexOf("MXXXX")!=-1)
{
    conversion = "MXL" + conversion.substring(conversion.indexOf("MXXXX")+5);
}

    if (conversion.indexOf("DXXXX")!=-1)
{
    conversion = "DXL" + conversion.substring(conversion.indexOf("DXXXX")+5);
}

    if (conversion.indexOf("CXXXX")!=-1)
{
    conversion = "CXL" + conversion.substring(conversion.indexOf("CXXXX")+5);

}
    if (conversion.indexOf("LXXXX")!=-1)
{
    conversion = "XC" + conversion.substring(conversion.indexOf("LXXXX")+5);
}

    if (conversion.indexOf("XXXX")!=-1)
{
    conversion = "XL" + conversion.substring(conversion.indexOf("XXXX")+4);

}
```



```
    if (conversion.indexOf("DIIII")!=-1)
{
    conversion = "DIV" + conversion.substring(conversion.indexOf("IIII")+5);
}

   if (conversion.indexOf("LIIII")!=-1)
{
    conversion = "LIV" + conversion.substring(conversion.indexOf("IIII")+5);
}

  if (conversion.indexOf("XIIII")!=-1)
{
    conversion = "XIV" + conversion.substring(conversion.indexOf("IIII")+5);
}

     if (conversion.indexOf("VIIII")!=-1)
{
    conversion = "IX" + conversion.substring(conversion.indexOf("IIII")+5);
}

 if (conversion.indexOf("IIII")!=-1)
{
    conversion = "IV" + conversion.substring(conversion.indexOf("IIII")+4);
}
```

TEST CASE 1: Running program through full execution 0-4000 FAIL



```
public static void main (String []args)
{
    for (int i=0; i<=4000; i++)
    {
        System.out.println("Decimal: " + i + " => " + decimalToRoman(i));

    }

}
```

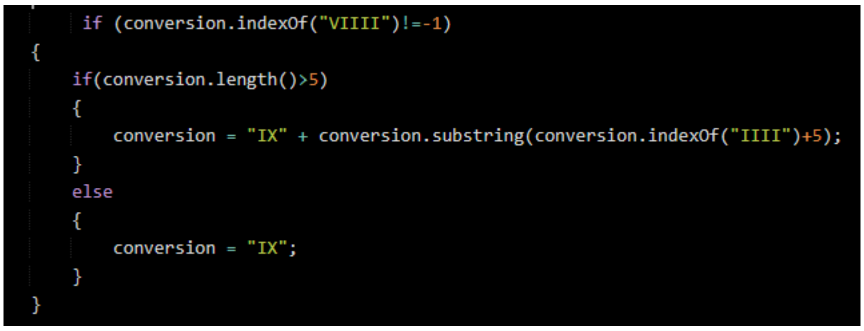
It has crashed at
Decimal: 0 =>

num is greater than: 1

Decimal: 1 => I

```
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 2 => II
num is greater than: 1
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 3 => III
num is greater than: 1
num is greater than: 1
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 4 => IV
num is greater than: 5
------------------------------
------------------------------
------------------------------
Decimal: 5 => V
num is greater than: 5
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 6 => VI
num is greater than: 5
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 7 => VII
num is greater than: 5
num is greater than: 1
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
------------------------------
Decimal: 8 => VIII
num is greater than: 5
num is greater than: 1
num is greater than: 1
num is greater than: 1
num is greater than: 1
------------------------------
------------------------------
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -1
        at java.base/java.lang.String.substring(String.java:1841)
        at Solution.decimalToRoman(Solution.java:234)
        at Solution.main(Solution.java:8)


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

I quickly believe following is remediation since there is no other characters in String beyond IX



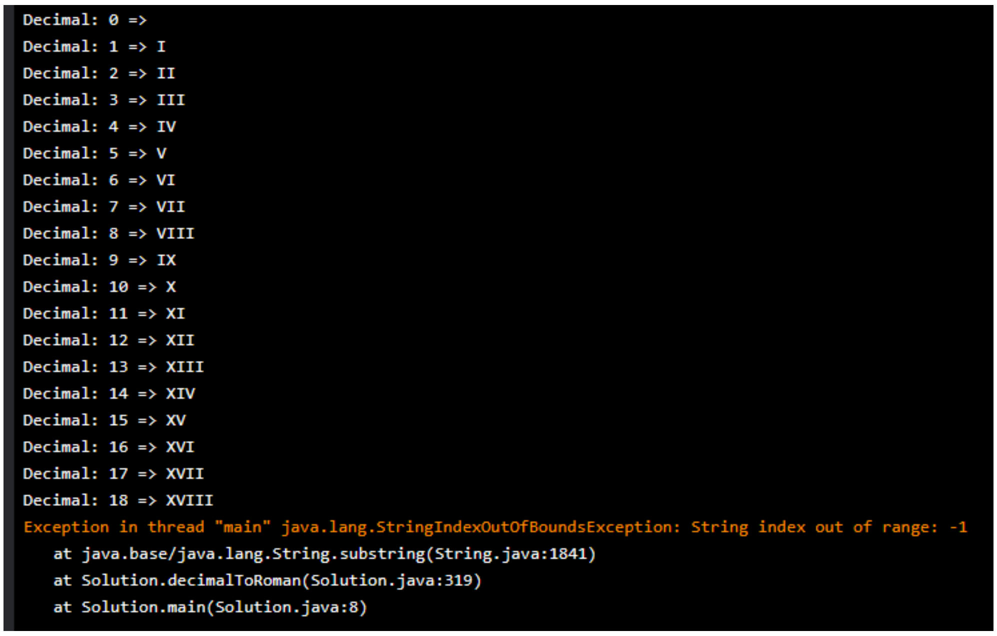
```
     if (conversion.indexOf("VIIII")!=-1)
{
    if(conversion.length()>5)
    {
        conversion = "IX" + conversion.substring(conversion.indexOf("IIII")+5);
    }
    else
    {
        conversion = "IX";
    }
}
```

TEST CASE 1a: Testing decimal 9   => PASS

I believe I need to apply the same logic across all the statements…

TEST CASE 2:   FAIL (issue at decimal 19)

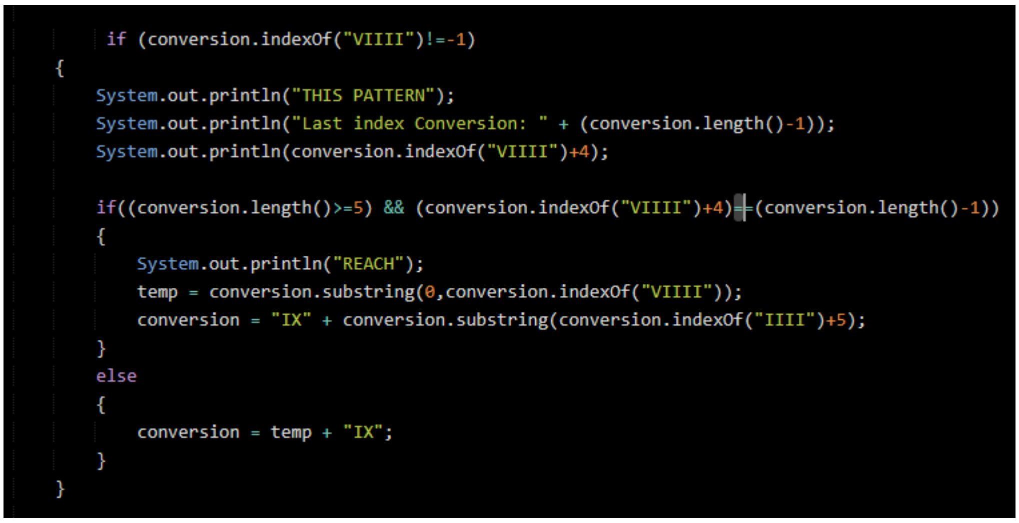
I have set up every loop identically



```
Decimal: 0 =>
Decimal: 1 => I
Decimal: 2 => II
Decimal: 3 => III
Decimal: 4 => IV
Decimal: 5 => V
Decimal: 6 => VI
Decimal: 7 => VII
Decimal: 8 => VIII
Decimal: 9 => IX
Decimal: 10 => X
Decimal: 11 => XI
Decimal: 12 => XII
Decimal: 13 => XIII
Decimal: 14 => XIV
Decimal: 15 => XV
Decimal: 16 => XVI
Decimal: 17 => XVII
Decimal: 18 => XVIII
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -1
   at java.base/java.lang.String.substring(String.java:1841)
   at Solution.decimalToRoman(Solution.java:319)
   at Solution.main(Solution.java:8)
```

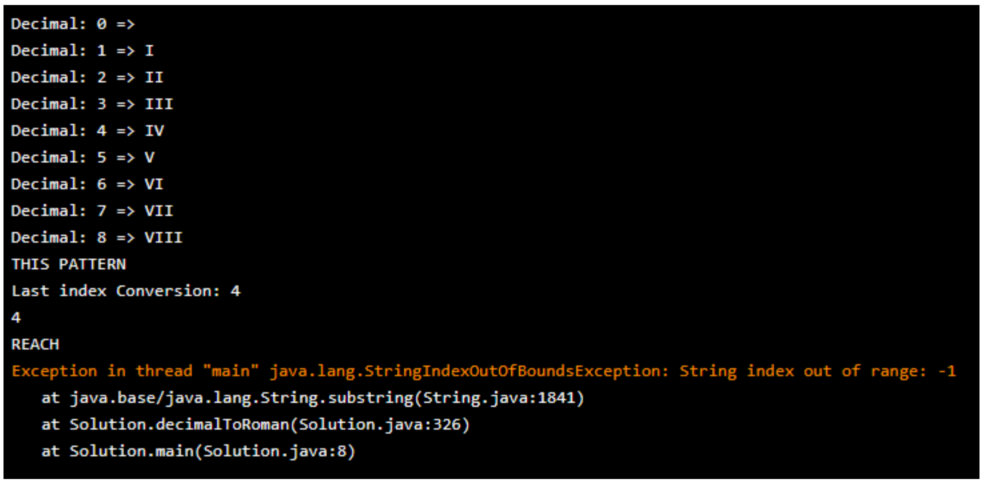
I progressed through two sets of codes..

The first one is as below:



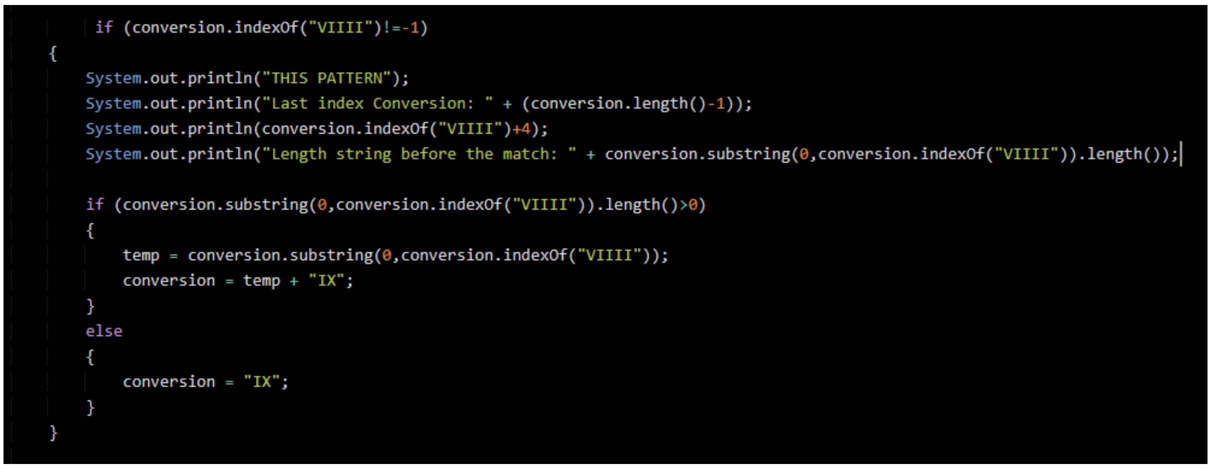
```java
        if (conversion.indexOf("VIIII")!=-1)
    {
        System.out.println("THIS PATTERN");
        System.out.println("Last index Conversion: " + (conversion.length()-1));
        System.out.println(conversion.indexOf("VIIII")+4);

        if((conversion.length()>=5) && (conversion.indexOf("VIIII")+4)==(conversion.length()-1))
        {
            System.out.println("REACH");
            temp = conversion.substring(0,conversion.indexOf("VIIII"));
            conversion = "IX" + conversion.substring(conversion.indexOf("IIII")+5);
        }
        else
        {
            conversion = temp + "IX";
        }
    }
```

However it rendered issue now on 9 (IX):



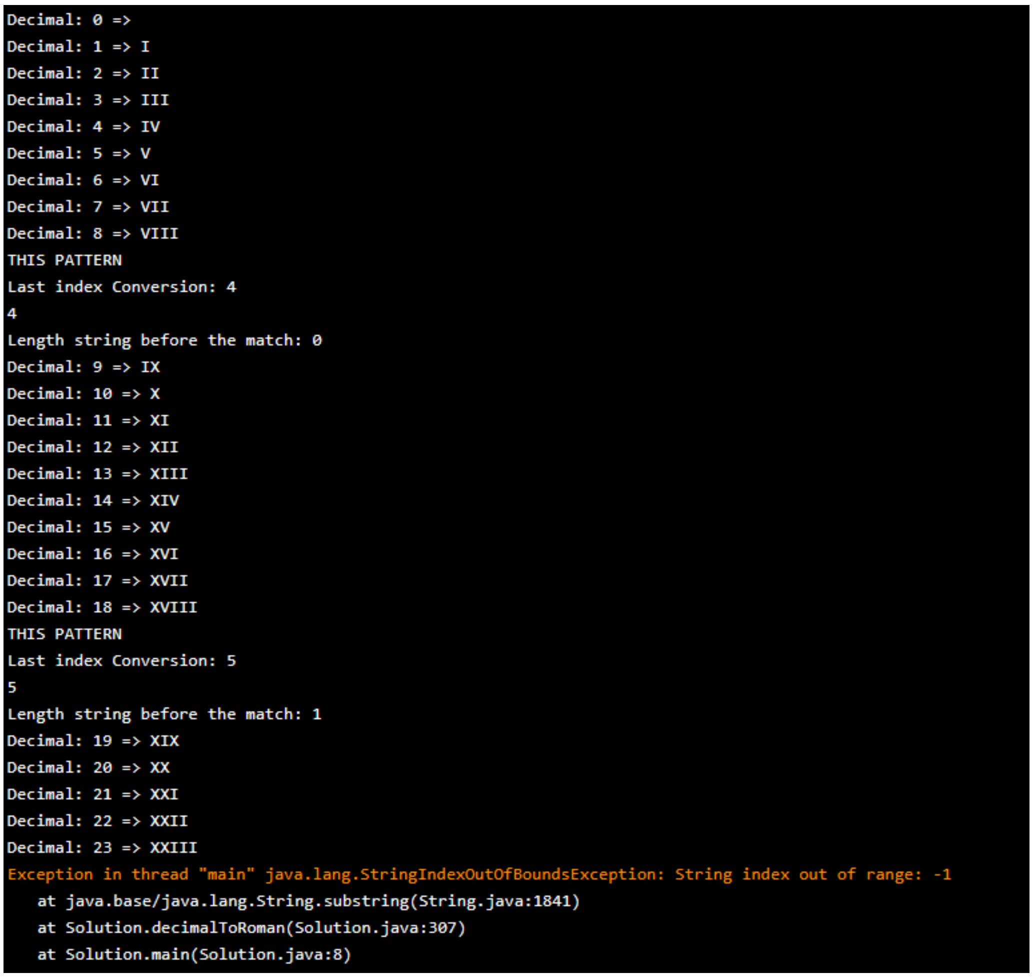
```
Decimal: 0 =>
Decimal: 1 => I
Decimal: 2 => II
Decimal: 3 => III
Decimal: 4 => IV
Decimal: 5 => V
Decimal: 6 => VI
Decimal: 7 => VII
Decimal: 8 => VIII
THIS PATTERN
Last index Conversion: 4
4
REACH
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -1
   at java.base/java.lang.String.substring(String.java:1841)
   at Solution.decimalToRoman(Solution.java:326)
   at Solution.main(Solution.java:8)
```

I used more understanding of the situation and adjusted the code to:



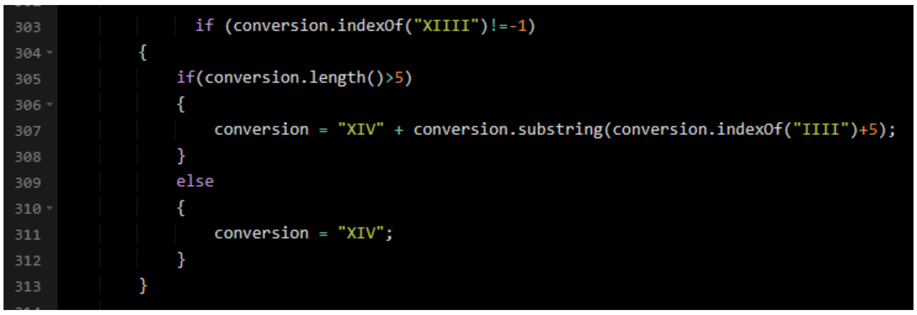
```java
        if (conversion.indexOf("VIIII")!=-1)
    {
        System.out.println("THIS PATTERN");
        System.out.println("Last index Conversion: " + (conversion.length()-1));
        System.out.println(conversion.indexOf("VIIII")+4);
        System.out.println("Length string before the match: " + conversion.substring(0,conversion.indexOf("VIIII")).length());

        if (conversion.substring(0,conversion.indexOf("VIIII")).length()>0)
        {
            temp = conversion.substring(0,conversion.indexOf("VIIII"));
            conversion = temp + "IX";
        }
        else
        {
            conversion = "IX";
        }
    }
```

It now passes 9 (IV) and 19 (XIX)



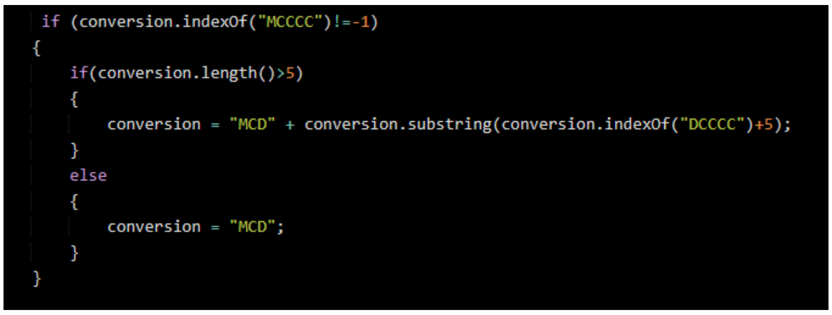
```
Decimal: 0 =>
Decimal: 1 => I
Decimal: 2 => II
Decimal: 3 => III
Decimal: 4 => IV
Decimal: 5 => V
Decimal: 6 => VI
Decimal: 7 => VII
Decimal: 8 => VIII
THIS PATTERN
Last index Conversion: 4
4
Length string before the match: 0
Decimal: 9 => IX
Decimal: 10 => X
Decimal: 11 => XI
Decimal: 12 => XII
Decimal: 13 => XIII
Decimal: 14 => XIV
Decimal: 15 => XV
Decimal: 16 => XVI
Decimal: 17 => XVII
Decimal: 18 => XVIII
THIS PATTERN
Last index Conversion: 5
5
Length string before the match: 1
Decimal: 19 => XIX
Decimal: 20 => XX
Decimal: 21 => XXI
Decimal: 22 => XXII
Decimal: 23 => XXIII
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: -1
   at java.base/java.lang.String.substring(String.java:1841)
   at Solution.decimalToRoman(Solution.java:307)
   at Solution.main(Solution.java:8)
```

I think I am now in a position that I need to apply the same statement logic into XIIII.



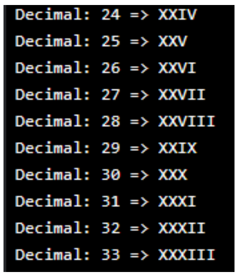
```
303            if (conversion.indexOf("XIIII")!=-1)
304        {
305            if(conversion.length()>5)
306            {
307                conversion = "XIV" + conversion.substring(conversion.indexOf("IIII")+5);
308            }
309            else
310            {
311                conversion = "XIV";
312            }
313        }
```

If it functions for this, I am all but certain I have tackled this challenge since I dealt with having any characters at the front in the conversion before the match (XIIII).
And since I have tackled the highest occurrence first (which is yet to appear in my testing), it should propagate the incorrect roman numeral correctly.



```
if (conversion.indexOf("MCCCC")!=-1)
{
    if(conversion.length()>5)
    {
        conversion = "MCD" + conversion.substring(conversion.indexOf("DCCCC")+5);
    }
    else
    {
        conversion = "MCD";
    }
}
```

TEST CASE 3: Modified code to resolve decimal 24 = PASS

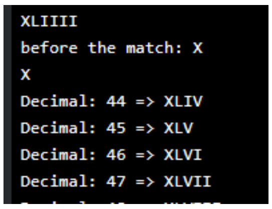


```
Decimal: 24 => XXIV
Decimal: 25 => XXV
Decimal: 26 => XXVI
Decimal: 27 => XXVII
Decimal: 28 => XXVIII
Decimal: 29 => XXIX
Decimal: 30 => XXX
Decimal: 31 => XXXI
Decimal: 32 => XXXII
Decimal: 33 => XXXIII
```

I can also see now that I now have to resolve LIIII
The good news is that I resolve issue for VIIII, XIIII and now LIIII
So it is moving sequentially through my code in the order I identified the statements.

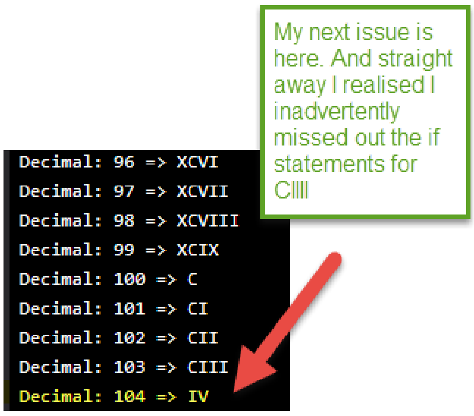
TEST CASE 4: Modified code to resolve decimal 44 = FAIL

I will first quickly check to ensure no coding errors



```
XLIIII
before the match: X
X
Decimal: 44 => XLIV
Decimal: 45 => XLV
Decimal: 46 => XLVI
Decimal: 47 => XLVII
```

It processes number all the way up to 403
It appears the first error is here…

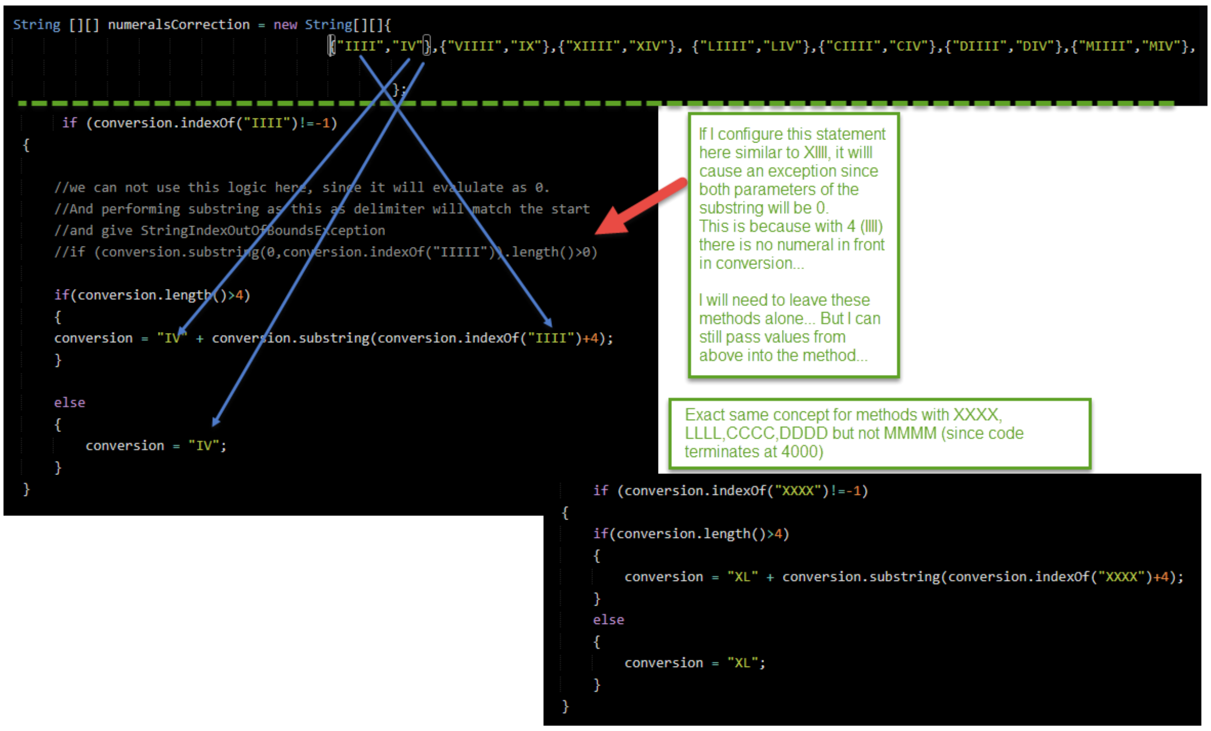


My next issue is here. And straight away I realised I inadvertently missed out the if statements for CIIII

```
Decimal: 96 => XCVI
Decimal: 97 => XCVII
Decimal: 98 => XCVIII
Decimal: 99 => XCIX
Decimal: 100 => C
Decimal: 101 => CI
Decimal: 102 => CII
Decimal: 103 => CIII
Decimal: 104 => IV
```

It is also appearing that I can potentially decrease the amount of code by passing parameters into the individual if statements since the code is repeat...
It will improve efficiency and also errors in coding.

But I see an issue already...
We know the first if statement to execute is. But I can see it can readily take the values from the String array I have devised.



```
String [][] numeralsCorrection = new String[][]{
                {"IIII","IV"},{"VIIII","IX"},{"XIIII","XIV"}, {"LIIII","LIV"},{"CIIII","CIV"},{"DIIII","DIV"},{"MIIII","MIV"},
                };
```

```
    if (conversion.indexOf("IIII")!=-1)
{

    //we can not use this logic here, since it will evalulate as 0.
    //And performing substring as this as delimiter will match the start
    //and give StringIndexOutOfBoundsException
    //if (conversion.substring(0,conversion.indexOf("IIIII")).length()>0)

    if(conversion.length()>4)
    {
    conversion = "IV" + conversion.substring(conversion.indexOf("IIII")+4);
    }

    else
    {
        conversion = "IV";
    }
}
```

If I configure this statement here similar to XIIII, it will cause an exception since both parameters of the substring will be 0.
This is because with 4 (IIII) there is no numeral in front in conversion...

I will need to leave these methods alone... But I can still pass values from above into the method...

Exact same concept for methods with XXXX, LLLL,CCCC,DDDD but not MMMM (since code terminates at 4000)

```
    if (conversion.indexOf("XXXX")!=-1)
{
    if(conversion.length()>4)
    {
        conversion = "XL" + conversion.substring(conversion.indexOf("XXXX")+4);
    }
    else
    {
        conversion = "XL";
    }
}
```

But before I implement this, I still feel its important I use my existing newly devised methods into test cases XXXX, LXXXX to ensure it functions before I can contemplate this universal solution.

<u>TEST CASE 4:   Amended all formulas and displaying output for 1-4000   FAIL</u>

I can errors have occurred below.
Since the output is too long, I have stopped the outputs after the error has occurred and in which I can recognise a pattern.

Decimal: 0 =>

Decimal: 1 => I

```
Decimal: 2 => II

Decimal: 3 => III

Decimal: 4 => IV

Decimal: 5 => V

Decimal: 6 => VI

Decimal: 7 => VII

Decimal: 8 => VIII

Decimal: 9 => IX

Decimal: 10 => X

Decimal: 11 => XI

Decimal: 12 => XII

Decimal: 13 => XIII

Decimal: 14 => XIV

Decimal: 15 => XV

Decimal: 16 => XVI

Decimal: 17 => XVII

Decimal: 18 => XVIII

Decimal: 19 => XIX

Decimal: 20 => XX

Decimal: 21 => XXI

Decimal: 22 => XXII

Decimal: 23 => XXIII

Decimal: 24 => XXIV

Decimal: 25 => XXV

Decimal: 26 => XXVI

Decimal: 27 => XXVII

Decimal: 28 => XXVIII

Decimal: 29 => XXIX
```

```
Decimal: 30 => XXX
Decimal: 31 => XXXI
Decimal: 32 => XXXII
Decimal: 33 => XXXIII
Decimal: 34 => XXXIV
Decimal: 35 => XXXV
Decimal: 36 => XXXVI
Decimal: 37 => XXXVII
Decimal: 38 => XXXVIII
Decimal: 39 => XXXIX
Decimal: 40 => XL
Decimal: 41 => XLI
Decimal: 42 => XLII
Decimal: 43 => XLIII
Decimal: 44 => XLIV
Decimal: 45 => XLV
Decimal: 46 => XLVI
Decimal: 47 => XLVII
Decimal: 48 => XLVIII
Decimal: 49 => XLIX
Decimal: 50 => L
Decimal: 51 => LI
Decimal: 52 => LII
Decimal: 53 => LIII
Decimal: 54 => LIV
Decimal: 55 => LV
Decimal: 56 => LVI
Decimal: 57 => LVII
```

```
Decimal: 58 => LVIII

Decimal: 59 => LIX

Decimal: 60 => LX

Decimal: 61 => LXI

Decimal: 62 => LXII

Decimal: 63 => LXIII

Decimal: 64 => LXIV

Decimal: 65 => LXV

Decimal: 66 => LXVI

Decimal: 67 => LXVII

Decimal: 68 => LXVIII

Decimal: 69 => LXIX

Decimal: 70 => LXX

Decimal: 71 => LXXI

Decimal: 72 => LXXII

Decimal: 73 => LXXIII

Decimal: 74 => LXXIV

Decimal: 75 => LXXV

Decimal: 76 => LXXVI

Decimal: 77 => LXXVII

Decimal: 78 => LXXVIII

Decimal: 79 => LXXIX

Decimal: 80 => LXXX

Decimal: 81 => LXXXI

Decimal: 82 => LXXXII

Decimal: 83 => LXXXIII

Decimal: 84 => LXXXIV

Decimal: 85 => LXXXV
```

```
Decimal: 86 => LXXXVI

Decimal: 87 => LXXXVII

Decimal: 88 => LXXXVIII

Decimal: 89 => LXXXIX

Decimal: 90 => XC

Decimal: 91 => XC

Decimal: 92 => XC

Decimal: 93 => XC

Decimal: 94 => XC

Decimal: 95 => XC

Decimal: 96 => XC

Decimal: 97 => XC

Decimal: 98 => XC

Decimal: 99 => XC

Decimal: 100 => C

Decimal: 101 => CI

Decimal: 102 => CII

Decimal: 103 => CIII

Decimal: 104 => CIV

Decimal: 105 => CV

Decimal: 106 => CVI

Decimal: 107 => CVII

Decimal: 108 => CVIII

Decimal: 109 => CIX

Decimal: 110 => CX

Decimal: 111 => CXI

Decimal: 112 => CXII

Decimal: 113 => CXIII
```

```
Decimal: 114 => CXIV

Decimal: 115 => CXV

Decimal: 116 => CXVI

Decimal: 117 => CXVII

Decimal: 118 => CXVIII

Decimal: 119 => CXIX

Decimal: 120 => CXX

Decimal: 121 => CXXI

Decimal: 122 => CXXII

Decimal: 123 => CXXIII

Decimal: 124 => CXXIV

Decimal: 125 => CXXV

Decimal: 126 => CXXVI

Decimal: 127 => CXXVII

Decimal: 128 => CXXVIII

Decimal: 129 => CXXIX

Decimal: 130 => CXXX

Decimal: 131 => CXXXI

Decimal: 132 => CXXXII

Decimal: 133 => CXXXIII

Decimal: 134 => CXXXIV

Decimal: 135 => CXXXV

Decimal: 136 => CXXXVI

Decimal: 137 => CXXXVII

Decimal: 138 => CXXXVIII

Decimal: 139 => CXXXIX

Decimal: 140 => CXL

Decimal: 141 => CXL
```

Decimal: 142 => CXL

Decimal: 143 => CXL

Decimal: 144 => CXL

Decimal: 145 => CXL

Decimal: 146 => CXL

Decimal: 147 => CXL

Decimal: 148 => CXL

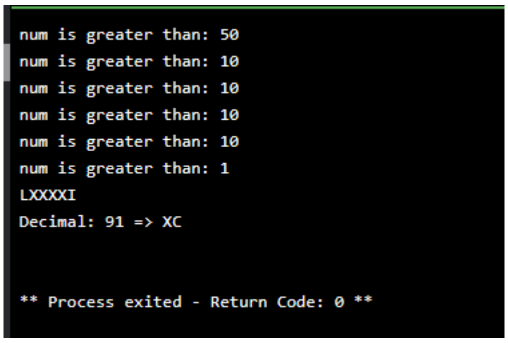
Decimal: 149 => CXL

Decimal: 150 => CL

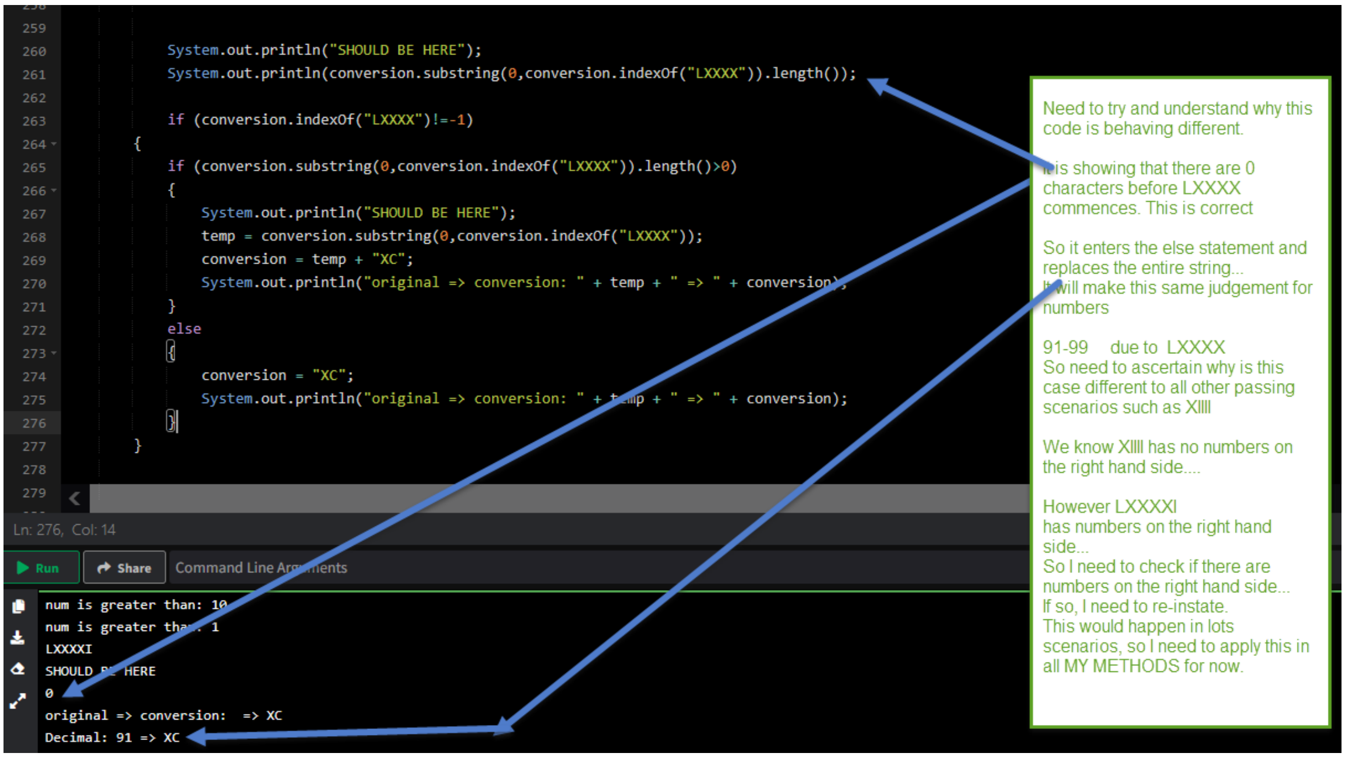
[91-99] is an example.
We know the incorrect numeral would have been LXXXXI
It has performed a translation to CX but not included the I at end.

TEST CASE 5: Understanding failed conversion for 91

I have enabled debugging, this tells me issue has happened in the methods once conversion string has been established…



```
num is greater than: 50
num is greater than: 10
num is greater than: 10
num is greater than: 10
num is greater than: 10
num is greater than: 1
LXXXXI
Decimal: 91 => XC


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



```
            System.out.println("SHOULD BE HERE");
            System.out.println(conversion.substring(0,conversion.indexOf("LXXXX")).length());

            if (conversion.indexOf("LXXXX")!=-1)
        {
            if (conversion.substring(0,conversion.indexOf("LXXXX")).length()>0)
            {
                System.out.println("SHOULD BE HERE");
                temp = conversion.substring(0,conversion.indexOf("LXXXX"));
                conversion = temp + "XC";
                System.out.println("original => conversion: " + temp + " => " + conversion);
            }
            else
            {
                conversion = "XC";
                System.out.println("original => conversion: " + temp + " => " + conversion);
            }
        }
```

Ln: 276, Col: 14

```
num is greater than: 10
num is greater than: 1
LXXXXI
SHOULD BE HERE
0
original => conversion:  => XC
Decimal: 91 => XC
```

Need to try and understand why this code is behaving different.

It is showing that there are 0 characters before LXXXX commences. This is correct

So it enters the else statement and replaces the entire string...
It will make this same judgement for numbers

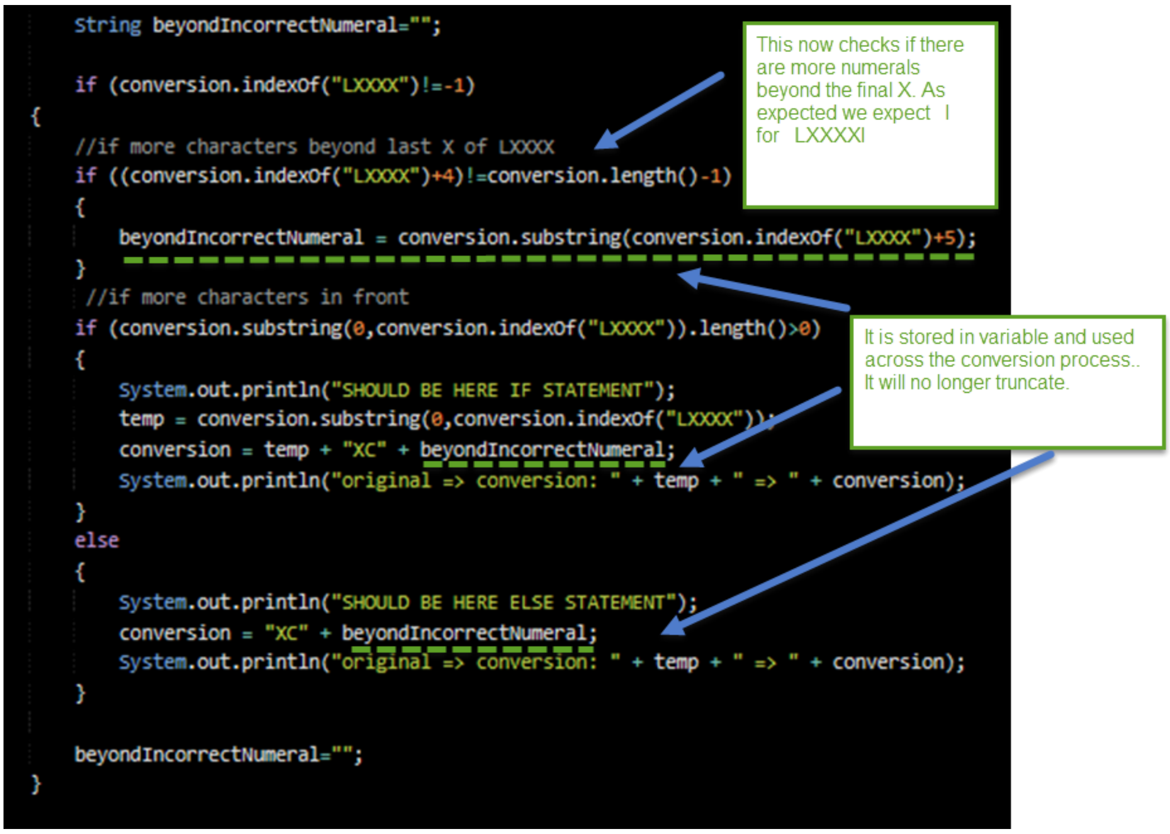
91-99 due to LXXXX
So need to ascertain why is this case different to all other passing scenarios such as XIII

We know XIII has no numbers on the right hand side....

However LXXXXI
has numbers on the right hand side...
So I need to check if there are numbers on the right hand side...
If so, I need to re-instate.
This would happen in lots scenarios, so I need to apply this in all MY METHODS for now.

I am also getting to my final objectives slower since I am refraining from using any exception handling.

<u>TEST CASE 6: Remediating issue with 91 by re-implementing the methods involved.</u>

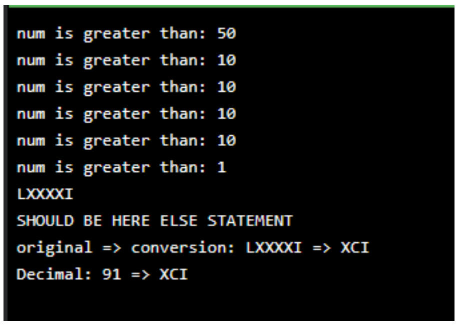


```
    String beyondIncorrectNumeral="";

    if (conversion.indexOf("LXXXX")!=-1)
{
    //if more characters beyond last X of LXXXX
    if ((conversion.indexOf("LXXXX")+4)!=conversion.length()-1)
    {
        beyondIncorrectNumeral = conversion.substring(conversion.indexOf("LXXXX")+5);
    }
    //if more characters in front
    if (conversion.substring(0,conversion.indexOf("LXXXX")).length()>0)
    {
        System.out.println("SHOULD BE HERE IF STATEMENT");
        temp = conversion.substring(0,conversion.indexOf("LXXXX"));
        conversion = temp + "XC" + beyondIncorrectNumeral;
        System.out.println("original => conversion: " + temp + " => " + conversion);
    }
    else
    {
        System.out.println("SHOULD BE HERE ELSE STATEMENT");
        conversion = "XC" + beyondIncorrectNumeral;
        System.out.println("original => conversion: " + temp + " => " + conversion);
    }

    beyondIncorrectNumeral="";
}
```

This now checks if there are more numerals beyond the final X. As expected we expect I for LXXXXI

It is stored in variable and used across the conversion process.. It will no longer truncate.

TEST CASE 6a Trying decimal 91 again   PASS



```
num is greater than: 50
num is greater than: 10
num is greater than: 10
num is greater than: 10
num is greater than: 10
num is greater than: 1
LXXXXI
SHOULD BE HERE ELSE STATEMENT
original => conversion: LXXXXI => XCI
Decimal: 91 => XCI
```

TEST CASE 7: Trying decimal 91 => 100   PASS

LXXXXI

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXI => XCI

Decimal: 91 => XCI

LXXXXII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXII => XCII

Decimal: 92 => XCII

LXXXXIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXIII => XCIII

Decimal: 93 => XCIII

LXXXXIIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXIIII => XCIIII

XCIIII

original => conversion: X => XCIV

Decimal: 94 => XCIV

LXXXXV

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXV => XCV

Decimal: 95 => XCV

LXXXXVI

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXVI => XCVI

Decimal: 96 => XCVI

LXXXXVII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXVII => XCVII

Decimal: 97 => XCVII

LXXXXVIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXVIII => XCVIII

Decimal: 98 => XCVIII

LXXXXVIIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: LXXXXVIIII => XCVIIII

original => conversion: XC => XCIX

Decimal: 99 => XCIX

C

Decimal: 100 => C

** Process exited - Return Code: 0 **

TEST CASE 8: Trying decimal 141 => 150
I am aware I now need to adjust method with CXXXX   PASS

CXXXXI

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXI => CXLI

Decimal: 141 => CXLI

CXXXXII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXII => CXLII

Decimal: 142 => CXLII

CXXXXIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXIII => CXLIII

Decimal: 143 => CXLIII

CXXXXIIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXIIII => CXLIIII

original => conversion: CX => CXLIV

Decimal: 144 => CXLIV

CXXXXV

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXV => CXLV

Decimal: 145 => CXLV

CXXXXVI

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXVI => CXLVI

Decimal: 146 => CXLVI

CXXXXVII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXVII => CXLVII

Decimal: 147 => CXLVII

CXXXXVIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXVIII => CXLVIII

Decimal: 148 => CXLVIII

CXXXXVIIII

SHOULD BE HERE ELSE STATEMENT

original => conversion: CXXXXVIIII => CXLVIIII

original => conversion: CXL => CXLIX

Decimal: 149 => CXLIX

CL

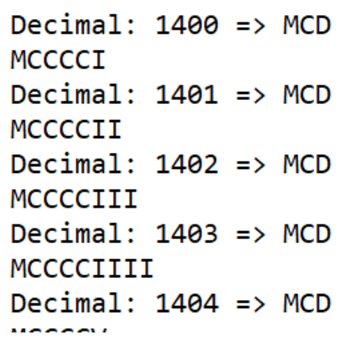
Decimal: 150 => CL

** Process exited - Return Code: 0 **

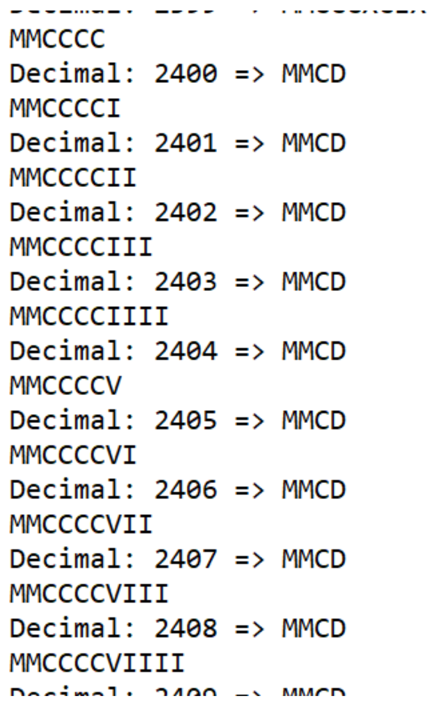
I am now confident I can implement a universal method now...
But just before this, I want to replicate the same logic across all scenarios and finally test 1-4000 again (decimal)

TEST CASE 9 Examining entire range 1-4000 FAIL

It can be seen that there are only two areas in which it has failed the conversion:



```
Decimal: 1400 => MCD
MCCCCI
Decimal: 1401 => MCD
MCCCCII
Decimal: 1402 => MCD
MCCCCIII
Decimal: 1403 => MCD
MCCCCIIII
Decimal: 1404 => MCD
```

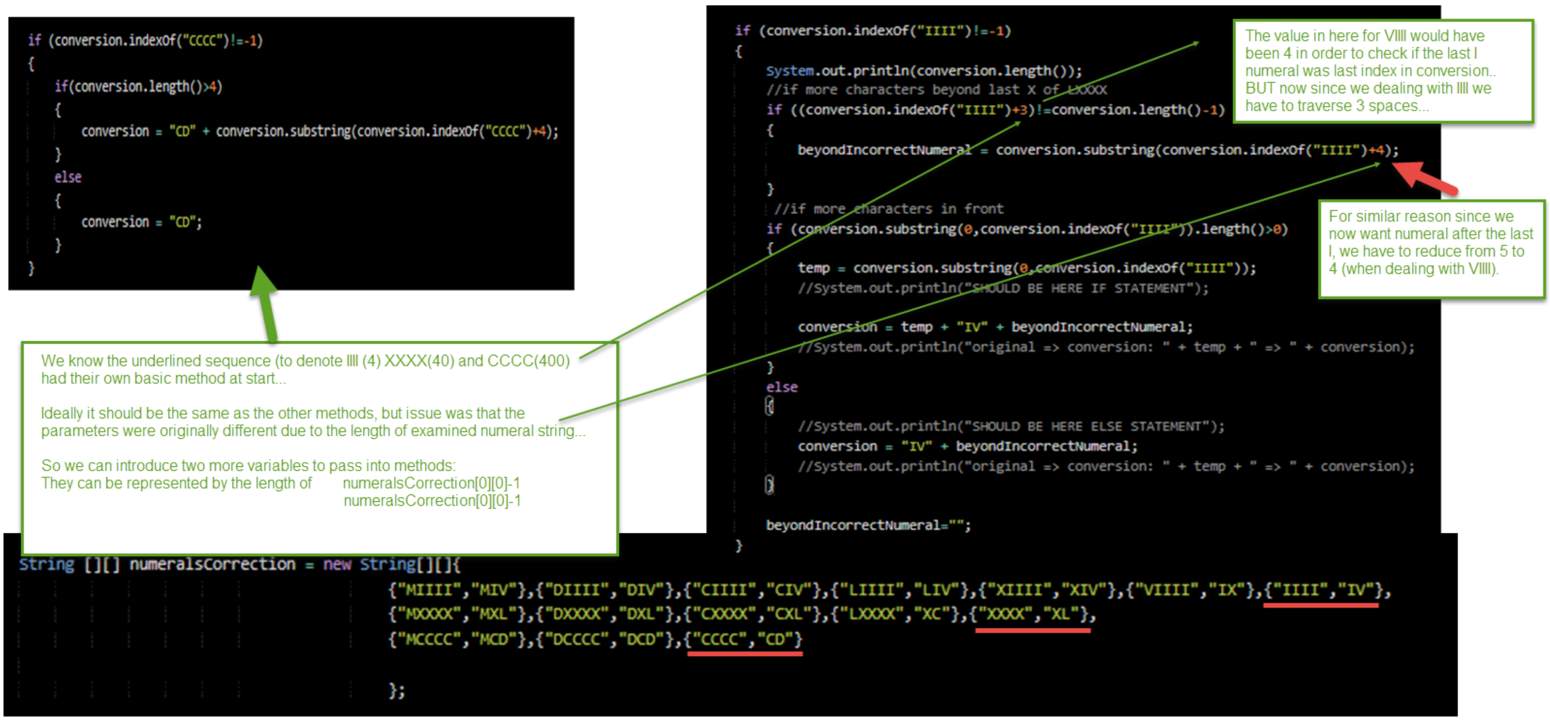


```
MMCCCC
Decimal: 2400 => MMCD
MMCCCCI
Decimal: 2401 => MMCD
MMCCCCII
Decimal: 2402 => MMCD
MMCCCCIII
Decimal: 2403 => MMCD
MMCCCCIIII
Decimal: 2404 => MMCD
MMCCCCV
Decimal: 2405 => MMCD
MMCCCCVI
Decimal: 2406 => MMCD
MMCCCCVII
Decimal: 2407 => MMCD
MMCCCCVIII
Decimal: 2408 => MMCD
MMCCCCVIIII
```

This more than suggests that it is an error in my coding.
The area of interest in both is CCCC
I have left this in now so that I can troubleshoot much quicker.
And it was realised that I had forgotten to overwrite the method with new implementation..

TEST CASE 8a Examining entire range 1-4000 PASS

I have now created the entire array so that I can pass parameters into the methods. I have to fine tune the code as below in order to consolidate for tidiness.



```
if (conversion.indexOf("CCCC")!=-1)
{
    if(conversion.length()>4)
    {
        conversion = "CD" + conversion.substring(conversion.indexOf("CCCC")+4);
    }
    else
    {
        conversion = "CD";
    }
}
```

```
if (conversion.indexOf("IIII")!=-1)
{
    System.out.println(conversion.length());
    //if more characters beyond last X of LXXXX
    if ((conversion.indexOf("IIII")+3)!=conversion.length()-1)
    {
        beyondIncorrectNumeral = conversion.substring(conversion.indexOf("IIII")+4);

    }
     //if more characters in front
    if (conversion.substring(0,conversion.indexOf("IIII")).length()>0)
    {
        temp = conversion.substring(0,conversion.indexOf("IIII"));
        //System.out.println("SHOULD BE HERE IF STATEMENT");

        conversion = temp + "IV" + beyondIncorrectNumeral;
        //System.out.println("original => conversion: " + temp + " => " + conversion);
    }
    else
    {
        //System.out.println("SHOULD BE HERE ELSE STATEMENT");
        conversion = "IV" + beyondIncorrectNumeral;
        //System.out.println("original => conversion: " + temp + " => " + conversion);
    }

    beyondIncorrectNumeral="";
}
```

The value in here for VIIII would have been 4 in order to check if the last I numeral was last index in conversion.. BUT now since we dealing with IIII we have to traverse 3 spaces...

For similar reason since we now want numeral after the last I, we have to reduce from 5 to 4 (when dealing with VIIII).

We know the underlined sequence (to denote IIII (4) XXXX(40) and CCCC(400) had their own basic method at start...

Ideally it should be the same as the other methods, but issue was that the parameters were originally different due to the length of examined numeral string...

So we can introduce two more variables to pass into methods:
They can be represented by the length of numeralsCorrection[0][0]-1
numeralsCorrection[0][0]-1

```
String [][] numeralsCorrection = new String[][]{
                    {"MIIII","MIV"},{"DIIII","DIV"},{"CIIII","CIV"},{"LIIII","LIV"},{"XIIII","XIV"},{"VIIII","IX"},{"IIII","IV"},
                    {"MXXXX","MXL"},{"DXXXX","DXL"},{"CXXXX","CXL"},{"LXXXX","XC"},{"XXXX","XL"},
                    {"MCCCC","MCD"},{"DCCCC","DCD"},{"CCCC","CD"}

                    };
```

TEST CASE 9:   Tidy up the code and check successful execution

PASS