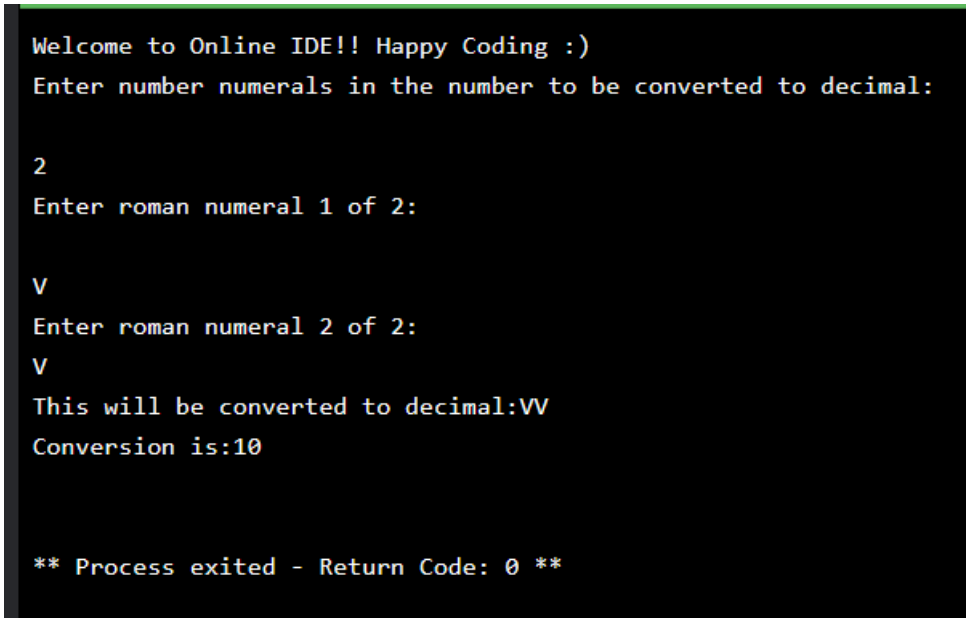
********** OUTPUT **********

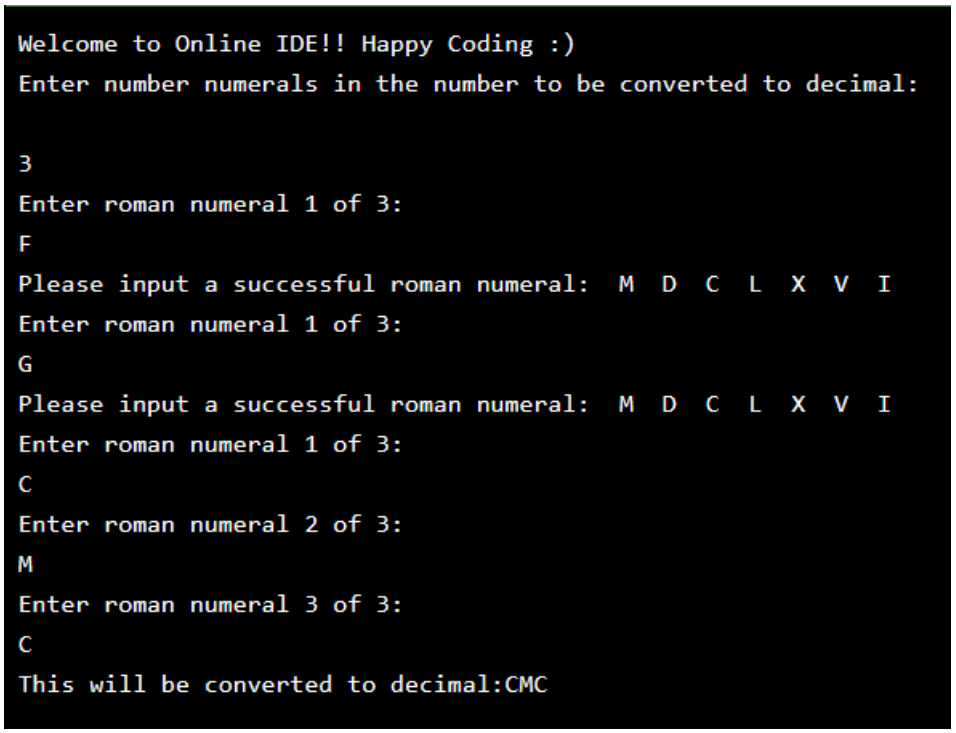
So far this is only a representation of getting the roman numeral entered by the end user and presenting this on the screen.
There is validation to ensure that correct numerals are presented.
Also, it will complete a conversion of a number consisting of single roman numeral.

Challenges appear in coding once it extends beyond single roman numeral.

The simple part is two identical numerals in a whole roman numeral such as XX, CC.



Example showing correct numeral but syntax incorrect.
This is to be explored further into code logic.



The following source online has pointed me towards all rules for a valid roman numeral.
Without having the correct representation, the conversion will be hindered.
I will tackle each rule, test and await outcome.

Note: I expect the solution to be dysfunctional once rules are applied. But as long rules begin validate, it's a huge starting point.
I am hoping all the rules are correct since it will impact end product.
https://www.cuemath.com/numbers/roman-numerals/

Rule 1: When certain numerals are repeated, the number represented by them is their sum. For example, II = 1 + 1 = 2, or XX = 10 + 10 = 20, or, XXX = 10 + 10 + 10 = 30.

```
Welcome to Online IDE!! Happy Coding :)
Enter number numerals in the number to be converted to decimal:

4
Enter roman numeral 1 of 4:

X
Enter roman numeral 2 of 4:
M
Enter roman numeral 3 of 4:
C
Enter roman numeral 4 of 4:
C
This will be converted to decimal:XMCC
numbers to be examined:4
sum so far: 200


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

```
Welcome to Online IDE!! Happy Coding :)
Enter number numerals in the number to be converted to decimal:
6
Enter roman numeral 1 of 6:
X
Enter roman numeral 2 of 6:
X
Enter roman numeral 3 of 6:
V
Enter roman numeral 4 of 6:
C
Enter roman numeral 5 of 6:
C
Enter roman numeral 6 of 6:
C
This will be converted to decimal:XXVCCC
numbers to be examined:6
sum so far: 20
sum so far: 220
does this happen
sum so far: 120
sum so far: 320
```

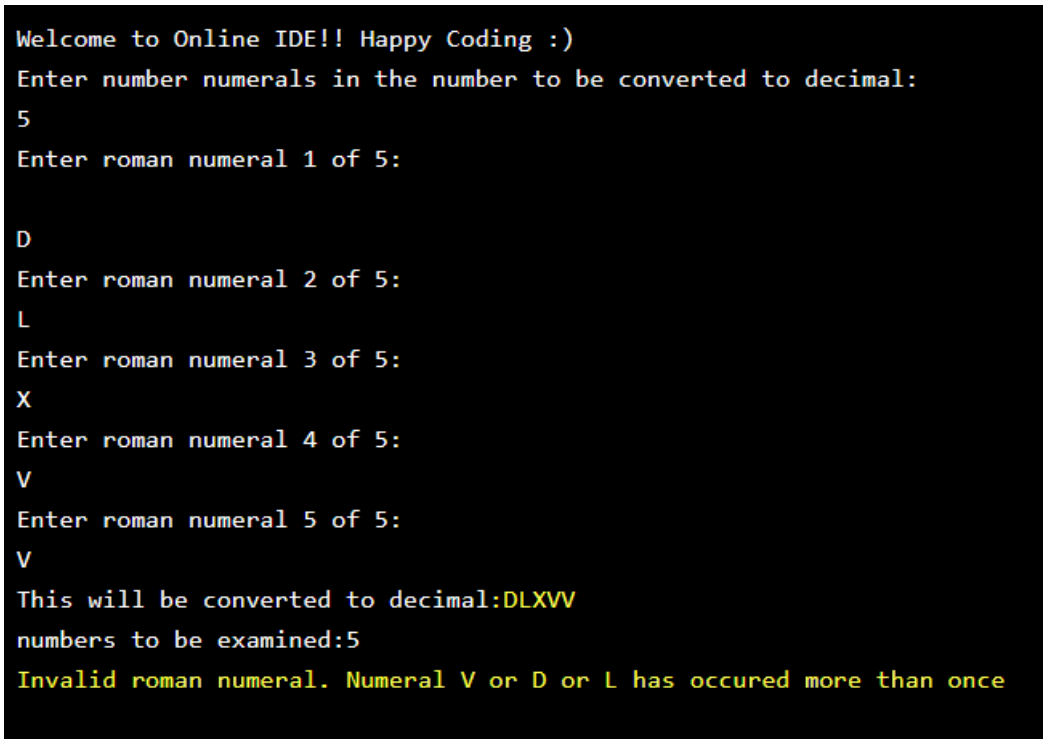
This has processed all adjacent numerals as rule 1.
XX = 20
CCC = 300
Total: 320

Rule 2: It is to be noted that no **Roman numerals can come together more than 3 times. For example, we cannot write 40 as XXXX**

```
Welcome to Online IDE!! Happy Coding :)
Enter number numerals in the number to be converted to decimal:

4
Enter roman numeral 1 of 4:

X
Enter roman numeral 2 of 4:

X
Enter roman numeral 3 of 4:

X
Enter roman numeral 4 of 4:
X
This will be converted to decimal:XXXX
numbers to be examined:4
There are more than three consecutive same characters: X
This is invalid roman numeral
```

Rule 3: The letters V, L, and D are not repeated

```
Welcome to Online IDE!! Happy Coding :)
Enter number numerals in the number to be converted to decimal:

3
Enter roman numeral 1 of 3:

V
Enter roman numeral 2 of 3:
V
Enter roman numeral 3 of 3:
V
This will be converted to decimal:VVV
numbers to be examined:3
sum so far: 10
does this happen
sum so far: 5
Invalid roman numeral. Numeral V or D or L has occured more than once
sum so far: 15
```



*** CODE ***

```
import java.util.Scanner;
import java.util.Arrays;
/*
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 :)");
```

```
      char[] acceptedNumerals = new char[]{'M','D','C','L','X', 'V','I'};
      //These are all accepted roman numerals

      int number; //This is set once end user specifies number numerals in their number to be inputted

      boolean noMatch=true;  // this flag is used to check if the numeral entered is a valid one

      int j;  //used to increment the character array containing user input of numeral
      int count=0;  // this keeps track number times end user has entered a character. It can be decremented if it
fails validation.

      //int pos;

      Scanner reader=null;


      // This is used to ascertain how many digits end user wants
      reader = new Scanner(System.in); // Reading from System.in
      System.out.println("Enter number numerals in the number to be converted to decimal:");
      number=reader.nextInt();

      char inputtedNumerals[] = new char[number];   // this creates character array with end user specified size
      int [][] posValues = new int [number][2];  // this is used to keep track of:

       // this is 2D array. 'number' simply increments with array index of the roman numeral in character array
      // the multidimensal aspect at 0 index will store:  index of which numeral i.e  M is 0, D is 1
                        //aspect at 1 index will store:  decimal conversion
      // This will support later analysis.


do
{
reader = new Scanner(System.in); // Reading from System.in
System.out.println("Enter roman numeral " + (count+1) + " of " + number + ":");   // end user prompted to enter
roman numeral
inputtedNumerals[count]=reader.next().charAt(0);
count++;

for (j=0; j<inputtedNumerals.length;j++)  // this will check each item in the inputted area against 7 roman
numerals.
                                 // the condition could equally be set to count for better loop efficiency if the
                                 //number entered is long
{

   for (int k=0; k<acceptedNumerals.length;k++)  // this goes through roman numerals
   {

   if (inputtedNumerals[j]==acceptedNumerals[k])  // this checks to see if the roman numerals entered are valid
   {

      posValues[count-1][0]=k;


      // This switch statement now checks 'k'validposition in hierarcy of the 7 numerals.
      //'M','D','C','L','X', 'V','I'

      switch(k)
      {
         case 0:
            posValues[count-1][1]=1000;
            break;

         case 1:
            posValues[count-1][1]=500;
```

```
            break;

         case 2:
            posValues[count-1][1]=100;
            break;

         case 3:
            posValues[count-1][1]=50;
            break;

         case 4:
            posValues[count-1][1]=10;
            break;

         case 5:
            posValues[count-1][1]=5;
            break;

         case 6:
            posValues[count-1][1]=1;
            break;
      }

      //pos=k;



       noMatch=false;   // This sets flag stating match has been found.
       k=acceptedNumerals.length-1;
       break;

       }

}
// This will now give information if the user input is valid for each numeral.
// If found to be incorrect, the count will go back and end user will need to enter it again.

 if (noMatch==true)
   {
     System.out.println("Please input a successful roman numeral:  M  D  C  L  X  V  I");
     count=count-1;
     break;
   }

}

} while (count<inputtedNumerals.length);

String str = String.valueOf(inputtedNumerals);
System.out.println("This will be converted to decimal:" + str);  // This informs end user of the roman numerals to
be converted

performConversion pc = new performConversion(acceptedNumerals, inputtedNumerals,posValues); // calls
constructor in class to perform conversion
}
}

class performConversion
{
   char[] inputtedNumerals;
   char[] acceptedNumerals;
   int [][] posValues;

    // This is the constructor used to call the converter method.
   public performConversion (char[] aNumerals, char [] inputNumerals, int [][] pValues)
```

```
  {
    this.acceptedNumerals=aNumerals;
    this.inputtedNumerals=inputNumerals;
    this.posValues=pValues;
    converter();
  }


  public void converter()
  {
    // There will be NO issues in performing conversion of a single roman numeral

    if (inputtedNumerals.length==1)    // This functions
    {
        System.out.println("Conversion is:" + posValues[0][1]);
    }

    // this is 2D array. count simply increments with array index of the roman numeral in character array
    // the multidimensal aspect at 0 index will store:  index of which numeral i.e  M is 0, D is 1
                //aspect at 1 index will store:  decimanl conversion


   // ONCE ALL PRINCIPLES ARE UNDERSTOOD FOR A VALID ROMAN NUMERAL, THE CODE WILL BE
IMPLEMENTED FURTHER

      if (inputtedNumerals.length==2)
    {
      if (posValues[0][0]==posValues[1][0])
      {

      System.out.println("Conversion is:" + (posValues[0][1] + posValues[1][1]));
      }

    }

  }


}
```

THIS IS MY CONTINUATION OF WORK UNDERTAKEN UP TO 7 April 2024

```
//best

import java.util.Scanner;
import java.util.Arrays;
/*
Online Java - IDE, Code Editor, Compiler
valu`va
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 :)");

      char[] acceptedNumerals = new char[]{'M','D','C','L','X', 'V','I'};
      //These are all accepted roman numerals

      int number; //This is set once end user specifies number numerals in their number to be inputted

      boolean noMatch=true;  // this flag is used to check if the numeral entered is a valid one

      int j;  //used to increment the character array containing user input of numeral
      int count=0;  // this keeps track number times end user has entered a character. It can be decremented if it
fails validation.

      Scanner reader=null;

      // This is used to ascertain how many digits end user wants
      reader = new Scanner(System.in); // Reading from System.in
      System.out.println("Enter number numerals in the number to be converted to decimal:");
      number=reader.nextInt();

      char inputtedNumerals[] = new char[number];   // this creates character array with end user specified size
      int [][] posValues = new int [number][2];  // this is used to keep track of:

       // this is 2D array. 'number' simply increments with array index of the roman numeral in character array
      // the multidimensal aspect at 0 index will store:  index of which numeral i.e  M is 0, D is 1
                  //aspect at 1 index will store:  decimal conversion
      // This will support later analysis.

      //all values will be set to 1 to ensure a repeat conversion is not processed;

do
{
reader = new Scanner(System.in); // Reading from System.in
System.out.println("Enter roman numeral " + (count+1) + " of " + number + ":");   // end user prompted to enter
roman numeral
inputtedNumerals[count]=reader.next().charAt(0);
count++;

for (j=0; j<inputtedNumerals.length;j++)  // this will check each item in the inputted area against 7 roman
numerals.
                              // the condition could equally be set to count for better loop efficiency if the
                              //number entered is long
{

   for (int k=0; k<acceptedNumerals.length;k++)  // this goes through roman numerals
   {

   if (inputtedNumerals[j]==acceptedNumerals[k])  // this checks to see if the roman numerals entered are valid
   {
```

```
      posValues[count-1][0]=k;        // Variable k keeps a record of index of the roman numeral stored in
acceptedNumerals

      // This switch statement now checks 'k'validposition in hierarcy of the 7 numerals.
      //'M','D','C','L','X', 'V','I'

      // Depending on the roman numeral value, it will also store the decimal conversion into posValues array.
      switch(k)
      {
         case 0:
            posValues[count-1][1]=1000;
            break;

         case 1:
            posValues[count-1][1]=500;
            break;

         case 2:
            posValues[count-1][1]=100;
            break;

         case 3:
            posValues[count-1][1]=50;
            break;

         case 4:
            posValues[count-1][1]=10;
            break;

         case 5:
            posValues[count-1][1]=5;
            break;

         case 6:
            posValues[count-1][1]=1;
            break;
      }


      noMatch=false;   // This sets flag stating match has been found.
      break;

      }

}
// This will now give information if the user input is valid for each numeral.
// If found to be incorrect, the count will go back and end user will need to enter it again.

 if (noMatch==true)
   {
     System.out.println("Please input a successful roman numeral:  M  D  C  L  X  V  I");
     count=count-1;
     break;
   }

}

} while (count<inputtedNumerals.length);  // do while loop will end once count reaches the input user for length
of roman numerals.


String str = String.valueOf(inputtedNumerals);
System.out.println("This will be converted to decimal:" + str);  // This informs end user of the roman numerals to
be converted
```

```
performConversion pc = new performConversion(acceptedNumerals, inputtedNumerals,posValues,str); // calls
constructor in class to perform conversion
}
}

class performConversion
{
   int sumNotAdded=0;
   int rule4;
   boolean rule3Fail=false;
   char[] inputtedNumerals;  // this contains numerals entered by end user

   char[] acceptedNumerals;  // this contains array of all accepted numerals
   int [][] posValues; // this keeps track for the  index for numeral and also its conversion into decimal
   int total;
   int sum;
   int consecutiveOccurrences;
   String numeralsToString;
   boolean correctOrder=false;

   int oneNumeral;
   int twoNumerals;
   boolean illegalSubtractive=false;
   int temp1=0;
   boolean valueBeforeSubtractiveNotation=false;

    // This is the constructor used to call the converter method.

   public performConversion (char[] aNumerals, char [] inputNumerals, int [][] pValues, String numeralsToString)
   {

      int pc[] = new int[pValues.length];
      this.acceptedNumerals=aNumerals;
      this.inputtedNumerals=inputNumerals;
      this.posValues=pValues;
      this.numeralsToString=numeralsToString;

      converter(pc);   // this variable has been passed into the method since for some reason it did not pick up the
variable even though scope was correct.
   }
   public void converter(int [] processedConversion)
   {
      // There will be NO issues in performing conversion of a single roman numeral
      System.out.println("the length is:" + inputtedNumerals.length);
      if (inputtedNumerals.length<2)    // This functions correctly.
      {
            System.out.println("Conversion is single:" + posValues[0][1]);

            processedConversion[0]=1;
            //oneNumeral = oneNumeral + posValues[0][1];
            System.exit(0);
      }

          // ONCE ALL PRINCIPLES ARE UNDERSTOOD FOR A VALID ROMAN NUMERAL, THE CODE WILL BE
IMPLEMENTED FURTHER


      System.out.println("numerals in the roman numeral:" + posValues.length);

      int valueConsecutiveOccurences[] = new int [posValues.length];
      int indexCount=0;

      // These values can only occur once in roman numeral. A counter is kept.
      int countV=0;
      int countL=0;
```

```
int countD=0;

int zeroIndexCount=0;  // on instances, the counter has to start from zeroindex notation

int rule4Single=0;  //this keeps a total of rule 4

int rule5=0;  // this keeps total at rule 5
int rule5total;
boolean rule5State=false;  // state is set if rule 5 is entered

int rule6=0;    // this keeps total at rule 6
int rule6total;
boolean rule6State=false;  // state is set if rule 6 is entered

int rule7=0;    // this keeps total at rule 7
int rule7total;
boolean rule7State=false;  // state is set if rule 7 is entered

// used for

 //Rule 4: Only I, X, and C can be used as subtractive numerals.
   //There can be 6 combinations when we subtract.
   //These are IV = 5 - 1 = 4; IX = 10 - 1 = 9; XL = 50 - 10 = 40;
   // XC = 100 - 10 = 90; CD = 500 - 100 = 400; and CM = 1000 - 100 = 900

   //Another useful IMPORTANT information found on a site is that two subtractive numerals can not be
adjacent such as:
   // CDCM since CM is higher however   CM CD is also invalid (range is 100-900)

   // The following is invalid:   IXIV even though IV is smaller than IX. This is because they are both in same
range 1-9
   // In that respect,   XC XL is invalid..  both notations are  10-90
   // XCIV is ok

   //{'M','D','C','L','X', 'V','I'};

   // I have partially implemented the rule here since it is too complex.

String IV =  "IV";
String IX =  "IX";
String XL =  "XL";
String XC =  "XC";
String CD=   "CD";
String CM=   "CM";

int threeInRow=0;

String [] subtractiveNumerals =  new String []{IV,IX,XL,XC,CD,CM};

int [][] subtractiveNumeralsRange = new int[6][2];

subtractiveNumeralsRange[0][0] = 1;   subtractiveNumeralsRange[0][1] = 4;
subtractiveNumeralsRange[1][0] = 1;   subtractiveNumeralsRange[1][1] = 9;
subtractiveNumeralsRange[2][0] = 10;   subtractiveNumeralsRange[2][1] = 40;
subtractiveNumeralsRange[3][0] = 10;   subtractiveNumeralsRange[3][1] = 90;
subtractiveNumeralsRange[4][0] = 100;   subtractiveNumeralsRange[4][1] = 500;
subtractiveNumeralsRange[5][0] = 100;   subtractiveNumeralsRange[5][1] = 900;


int sum1=0;
int total1=0;
boolean flagError=false;

boolean result=false;
int total2=0;
```

```
int overallTotal=0;
boolean twoSubtractiveNotations=false;
boolean singleSubtractiveNotations=false;
boolean IplacedIncorrect=false;



boolean rule2State=false;
int tempBefore=0;

int rule2=0;
int rule2total=0;
boolean invalidSubtractiveNumeral=false;
int temp=0;

boolean VLDcheck=false;

boolean illegalThreeInARowState=false;
int rangeLower=0;
int rangeUpper=0;

String checkComplete[] = new String[subtractiveNumerals.length];

//----------------------------------------

// This has been completed outside main loop because it is processing a string to check values
// this needs to go inside main loop. otherwise can not check the index of posValues

System.out.println("In rule 4");
for (int i=0; i<subtractiveNumerals.length; i++)   // this is going through all 6 combos IV, IX, XL.....
{
   for (int j=0; j<subtractiveNumerals.length; j++ )   // this is going through all 6 combos IV, IX, XL.....
   {
      // NEED MORE LOGIC TO PREVENT IT FROM COMPARING AGAINST ITSELF.
      // OTHERWISE IT WILL PREVENT SINGLE OCCURENCE OF SUBTRACTIVE NOTATION FROM BEING
PROCESSED

      if (j==i)
      {
         continue;
      }

      // this ensures that there are matches against two subtractive notations before processing
      if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 &&
numeralsToString.indexOf(subtractiveNumerals[j])!=-1)
      {
          twoSubtractiveNotations = true;

         // this checks if subtractive notation lower value appears at index before higher subtractive notation

         if (numeralsToString.indexOf(subtractiveNumerals[i]) <
numeralsToString.indexOf(subtractiveNumerals[j]))

         {
            checkComplete[i]=subtractiveNumerals[i]+subtractiveNumerals[j];
            if(checkComplete[i].indexOf(subtractiveNumerals[j])==-1)
            {




            // THERE IS ISSUE IN THIS PART OF CODE:
            // FOR INSTANCE IF IT DETECTS   M CM IV    It will first find circumstance ok since  IV is in i loop and
CM is in j loop
```

```
                //and hence since index of IV will appear higher, it will decide not to proceed with this.
                // HOWEVER as the i loop progresses  it will become CM in i loop.. It will then find  IV in the j loop.
                // In this instance CM will appear lower in the index notation... hence the loop will validate as true and proceed.
                // Need a way to stop this from occuring.

                System.out.println("WHY");
                System.out.println(numeralsToString.indexOf(subtractiveNumerals[i]));
                System.out.println((subtractiveNumerals[i]));

                System.out.println(numeralsToString.indexOf(subtractiveNumerals[j]));
                System.out.println((subtractiveNumerals[j]));


                System.out.println(numeralsToString + " is invalid");
                System.out.println("Incorrect roman numeral. " +  subtractiveNumerals[i] + " portion should be before: " + subtractiveNumerals[j]);
                System.out.println("A group of numerals written in subtractive notation, of lower value, " + subtractiveNumerals[i] + " ( =" + subtractiveNumeralsRange[i][1]+"), cannot precede another group of numerals written in subtractive notation," + subtractiveNumerals[j] + " ( =" + subtractiveNumeralsRange[j][1]+").");

                illegalSubtractive=true;
                 //
                 invalidSubtractiveNumeral=true;
                  processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
                  temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
                  processedConversion[temp]=1;

              }

              }


          // this now checks the subtractive notations in correct order

          if (numeralsToString.indexOf(subtractiveNumerals[i]) > numeralsToString.indexOf(subtractiveNumerals[j]))
          {
             //Both these subtractive numerals appear. a case of now checking if range is same

          System.out.println("This should print out start of the range:" + subtractiveNumeralsRange[i][0]);
          System.out.println("This should print out end of the range:" + subtractiveNumeralsRange[i][1]);
          System.out.println("This should print out start of the range:" + subtractiveNumeralsRange[j][0]);
          System.out.println("This should print out end of the range:" + subtractiveNumeralsRange[j][1]);

             if (subtractiveNumeralsRange[i][0] == subtractiveNumeralsRange[j][0])  // these subtractive notations are in the same range
             {
                System.out.println(subtractiveNumerals[i] + " is in the range: " + subtractiveNumeralsRange[i][0] + " - " + subtractiveNumeralsRange[i][1]);
                System.out.println(subtractiveNumerals[j] + " is in the range: " + subtractiveNumeralsRange[j][0] + " - " + subtractiveNumeralsRange[j][1]);
                System.out.println("They are both in the same range");


                System.out.println("not sure whats here:" + subtractiveNumerals[i] + "  "+ numeralsToString.indexOf(subtractiveNumerals[i]));
                System.out.println("not sure whats here:" + subtractiveNumerals[j] + "  "+ numeralsToString.indexOf(subtractiveNumerals[i+1]));

                System.out.println("not sure whats here:" + numeralsToString.indexOf(subtractiveNumerals[j]));
                 System.out.println("not sure whats here:" + numeralsToString.indexOf(subtractiveNumerals[j]+1));


                processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
```

```
                temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
                processedConversion[temp]=1;
                processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]=1;
                temp = numeralsToString.indexOf(subtractiveNumerals[j]) + 1;
                processedConversion[temp]=1;

                rule4=0;

            }

            if (subtractiveNumeralsRange[i][0] != subtractiveNumeralsRange[j][0])  // this ensures that subtractive
notations are not in the same range
            {
                //System.out.println("What is here: " +
processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]);
                //System.out.println("What is here: " +
processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]);

                if(processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]!=1 &&
processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]!=1 )
                {

            if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 &&
numeralsToString.indexOf(subtractiveNumerals[j])!=-1)
            {
                System.out.println("correct order");
                correctOrder=true;

                processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
                //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])][1]]);

                temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
                processedConversion[temp]=1;
                //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])+1][1]]);

                switch (subtractiveNumerals[i])
                {
                   case "IV":
                      rule4 = rule4 + 4;
                      break;

                   case "IX":
                      rule4 = rule4 + 9;
                      break;

                   case "XL":
                      rule4 = rule4 + 40;
                      break;

                   case "XC":
                      rule4 = rule4 + 90;
                      break;

                   case "CD":
                      rule4 = rule4 + 400;
                      break;

                   case "CM":
                      rule4 = rule4 + 900;
                      break;

                   default:
                   invalidSubtractiveNumeral = true;
```

```
                }

                switch (subtractiveNumerals[j])
                {
                   case "IV":
                      rule4 = rule4 + 4;
                      break;

                   case "IX":
                      rule4 = rule4 + 9;
                      break;

                   case "XL":
                      rule4 = rule4 + 40;
                      break;

                   case "XC":
                      rule4 = rule4 + 90;
                      break;

                   case "CD":
                      rule4 = rule4 + 400;
                      break;

                   case "CM":
                      rule4 = rule4 + 900;
                      break;

                   default:
                   invalidSubtractiveNumeral = true;
                }

              }
              }
            }

            }
            }
        }

        boolean correctOrder=false;
        System.out.println("total after rule4:" + rule4);
        // this confirms if a match is found.  -1 value is no match. 0 is a match.
        System.out.println(subtractiveNumerals[i] + "   " + numeralsToString.indexOf(subtractiveNumerals[i]));

        // this now is just simply going through all subtractive notations if there is no other existing
        // this can not run if the above circumstance is true.

           if (!twoSubtractiveNotations)
        {
           System.out.println("BE HERE!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
           if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1)  // This double checks that there is only one
subtractive notation
           {System.out.println("BE HERE AGAIN!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
              if (processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]!=1)
              {
              System.out.println("BE HERE AGAIN2!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
               // **This is now checking if X for instance is occuring in the numerals and also subtractive notations
              // ** for instance   XCX

              //the code has not referenced range at all.. this will cause code to fail in scenarios such as MCM
where it is VALID

      //subtractiveNumeralsRange[0][0] = 1;   subtractiveNumeralsRange[0][1] = 4;
      //subtractiveNumeralsRange[1][0] = 1;   subtractiveNumeralsRange[1][1] = 9;
```

```
//subtractiveNumeralsRange[2][0] = 10;   subtractiveNumeralsRange[2][1] = 40;
//subtractiveNumeralsRange[3][0] = 10;   subtractiveNumeralsRange[3][1] = 90;
//subtractiveNumeralsRange[4][0] = 100;   subtractiveNumeralsRange[4][1] = 500;
//subtractiveNumeralsRange[5][0] = 100;   subtractiveNumeralsRange[5][1] = 900;

//String [] subtractiveNumerals =  new String []{IV,IX,XL,XC,CD,CM};

        System.out.println("ASDASDBGDFSDGFDGFDG");
        System.out.println(subtractiveNumerals[i]);

        if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 && posValues.length>2)
        // if it finds a match of subtractive notation in the numeral consisting of 3 or more numerals
        {
        System.out.println(numeralsToString.indexOf(subtractiveNumerals[i]));

        // this ensures if numeral appears after notation, it gets position 2
        if (numeralsToString.indexOf(subtractiveNumerals[i])==0)
        {
           temp = numeralsToString.indexOf(subtractiveNumerals[i])+2;
           System.out.println("NUMERAL AFTER NOTATION!");
        }

        // this ensures if numeral appears before notation, it gets position 0
        if (numeralsToString.indexOf(subtractiveNumerals[i])==1)
        {
           temp = numeralsToString.indexOf(subtractiveNumerals[i])-1;
           valueBeforeSubtractiveNotation = true;
           tempBefore= numeralsToString.indexOf(subtractiveNumerals[i]);
        }

        //temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 2;  // sets temp to allow comparison 1st
with 3rd char
        System.out.println("vcbvbnm,,");
        System.out.println(posValues.length-1);
        System.out.println(numeralsToString.indexOf(subtractiveNumerals[i]));


        // NEED MORE LOGIC HERE TO CHANGE HOW TEMP IS GENERATED
        // FOR INSTANCE WITH XCM  temp is one places before than location of index notation


         //HOWEVER WITH  MCM   the M is one place before subtractive notation
         //So need to check in both directions

         System.out.println("first");
        }

        // this is length of two inputtedNumerals. It has no bearing on above.
        if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 && posValues.length==2) //this should
allow normal flow of app?
        {
           temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
           System.out.println("second");
        }

         System.out.println("temp is currrentl: " + temp);
         System.out.println("WWWWWWWWWWWWWWWWWWWWWWWWW");

         if (temp<=posValues.length && posValues.length>2 )
         // this ensures that there is a value in numeral to compare against
         // it is set to minumum 3 numerals

        {  //This checks   XCX    first char of subtractivenotation X   against  2 positions in main array

           //System.out.println("B:" + numeralsToString.charAt(temp-1));
```

```
                //now need to know the range of the subtractive notation and also the non notation
                // which is either before or after subtractive notation
                // VERY IMPORTANT

                //since the code does not include array holding the numeral and its decimal value, switch
statement
                // will need to be used. posValues was declared two dimensional holding.
                //Also since something like XC, the index will be split in two and lower and upper range will be
outputted.

                //range for subtractive notation

                switch (subtractiveNumerals[i].charAt(0))
                {
                   case 'I':
                        rangeLower=1;
                        break;

                  case 'V':
                     rangeLower=5;
                        break;

                  case 'X':
                     rangeLower=10;
                        break;

                  case 'L':
                     rangeLower=50;
                        break;

                  case 'C':
                     rangeLower=100;
                        break;

                  case 'D':
                     rangeLower=500;
                        break;

                  case 'M':
                     rangeLower=1000;
                        break;

                  default:
                  System.out.println("Do not reach here");

                }

                switch (subtractiveNumerals[i].charAt(1))
                {
                   case 'I':
                        rangeUpper=1;
                        break;

                  case 'V':
                     rangeUpper=5;
                        break;

                  case 'X':
                     rangeUpper=10;
                        break;

                  case 'L':
                     rangeUpper=50;
                        break;
```

```
                case 'C':
                   rangeUpper=100;
                      break;

                case 'D':
                   rangeUpper=500;
                      break;

                case 'M':
                   rangeUpper=1000;
                      break;

                default:
                System.out.println("Do not reach here");

               }

               System.out.println(subtractiveNumerals[i].charAt(0));
               //System.out.println(numeralsToString.charAt(temp));


               // This is now the example for XC   X   X=10   range for XC (is  10-90 )
               // So evaluating that value at location is not greater or equal to 10

               System.out.println("IVC why not rule 2");

               System.out.println(tempBefore);
               System.out.println(rangeUpper);
               System.out.println(rangeLower);
               System.out.println(zeroIndexCount);
               System.out.println(temp);
               //System.out.println(posValues[tempBefore+2][1]);


               //if (tempBefore>0)  // this is showing there is check backwards
               //examples would be CCM
            //{
            if (/*subtractiveNumerals[i].charAt(0) == numeralsToString.charAt(tempBefore)  &&*/
posValues[temp][1] <(subtractiveNumeralsRange[i][1]) && tempBefore>temp)
            {
               System.out.println("%*%*%*");

               System.out.println("2. A numeral (a letter) of lower value" + numeralsToString.charAt(tempBefore-
1) +  "(=" + posValues[tempBefore-1][1] + ") "+  "cannot precede a group of numerals written in subtractive
notation,");
               System.out.println(subtractiveNumerals[i] + "( ="+ subtractiveNumeralsRange[i][1] +").");

               //C   CM: A numeral (a letter) of lower value, C ( = 100), cannot precede a group of
               //numerals written in subtractive notation, CM ( = 900).

               illegalSubtractive=true;


            }

            //}
            System.out.println("Rule 1..... values");
            System.out.println(posValues[zeroIndexCount][1]);
            System.out.println(rangeLower);
            System.out.println(rangeUpper);
            System.out.println(tempBefore);
            System.out.println(temp);
            System.out.println("What is I: " +i);
```

```
                //temp is set +2 from index of start subtractive notation
                  if (posValues[temp][1] >=subtractiveNumeralsRange[i][0] && posValues[temp][1]
<=subtractiveNumeralsRange[i][1] && temp>tempBefore)  /*&& posValues.length>2*/
              {
                 System.out.println("%*%*%*");

                 System.out.println("1. The numeral " + numeralsToString.charAt(temp) + "(=" + posValues[temp][1]
+ ") " + " cannot be placed after a group of numerals written in subtractive notation: " + subtractiveNumerals[i]);
                 System.out.println("Of the same range: " +  "( = " + subtractiveNumeralsRange[i][0] + "-" +
subtractiveNumeralsRange[i][1] + ").");

                 //The numeral X ( = 10) cannot be placed after a group of numerals written in subtractive notation,
XC ( = 90),
                 //of the same range value (10 - 90)

                 illegalSubtractive=true;
              }

              // this is now using scenario such as C CM  where  C (100)  appears before    CM(subtractive numeral) -
range (100-900)
              // C can not be lower than 900(upper range).  It can be equal for scenarios such as MCM (this is valid
1900).

              System.out.println("TempBefore:" + tempBefore);


              // This is using IV C as an EXAMPLE

              System.out.println(subtractiveNumeralsRange[i][1]);
              //System.out.println(posValues[temp][1]);


              //1 has been subtracted from posValues since it is zero index based

              //This in the if loop will ensure instances such as M CM are not trapped
              //However C CM is trapped

              // RangeUpper!= posValues[temp-1][1]


              System.out.println("this location----------");
              System.out.println(subtractiveNumeralsRange[i][1]);
              //System.out.println(posValues[temp-1][1]);
              System.out.println(tempBefore);
              System.out.println(temp);
              System.out.println(zeroIndexCount);
              System.out.println("YOU ARE NOW HERE**********************");

              System.out.println(subtractiveNumeralsRange[i][1]);
              System.out.println(posValues[temp][1]);
              System.out.println(temp);
              System.out.println(tempBefore);



               if (subtractiveNumeralsRange[i][1] < posValues[temp][1] && tempBefore<temp)  /*&&
posValues.length>2*/
              {
                 System.out.println("%*%*%*");

              System.out.println("3. A group of numerals written in subtractive notation, of lower value,");
              System.out.println(subtractiveNumerals[i] + " ( = " + (rangeUpper-rangeLower) + "), cannot precede a
numeral of larger value, "   +  numeralsToString.charAt(temp) + "( = "+posValues[temp][1]+".");
```

```
            //IV  C: A group of numerals written in subtractive notation, of lower value, IV ( = 4),
            //cannot precede a numeral of larger value, C ( = 100)

               //System.out.println("1. The numeral " + numeralsToString.charAt(temp) + "(=" +
posValues[temp][1] + ") " + " cannot be placed after a group of numerals written in subtractive notation: " +
subtractiveNumerals[i]);
               //System.out.println("Of the same range: " +  "( = " + rangeLower + "-" + rangeUpper + ").");

               illegalSubtractive=true;
            }



            // THIS CODE IS REQUIRED FOR EXAMPLE SUCH AS IVC


           // this is now checking if such a sequence exists CMM. This is not legal since
           //subtractive notation CM (900) is less than M (1000).

               switch (subtractiveNumerals[i])
               {
                  case "IV":
                     rule4Single = rule4Single + 4;
                     break;

                  case "IX":
                     rule4Single = rule4Single + 9;
                     break;

                  case "XL":
                     rule4Single = rule4Single + 40;
                     break;

                  case "XC":
                     rule4Single = rule4Single + 90;
                     break;

                  case "CD":
                     rule4Single = rule4Single + 400;
                     break;

                  case "CM":
                     rule4Single = rule4Single + 900;
                     break;

                  default:
                  invalidSubtractiveNumeral = true;
               }

               System.out.println("Does it even get here");



               //System.out.println("This should now print out instances like: " + numeralsToString);
               System.out.println("It's conversion is:" +rule4Single);
               twoSubtractiveNotations=false;
               singleSubtractiveNotations=true;
            }


         processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
            temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
            processedConversion[temp]=1;
```

```
        }
          }


        }
    }

    //-----------------------------------------------------------------

    //}
  //}
    //  MAIN PROGRAM EXECUTION
    //this will examine decimal number stored. If identical numerals in consecutive it will add to the total.

    for (int m=1; m<posValues.length; m++)  // this is starting not at zero index since compared to previous value
    {
     if (processedConversion[m]!=1)
       {
        //posValues[m][0]=numeral position     //[m][1]=decimal
       // This section is best completed examining string of the character array of inputted numerals.

       // Rule 3: The letters V, L, and D are not repeated.
       // There are acceptedNumerals[1], acceptedNumerals[3], acceptedNumerals[5]
       // posValues[m][0] stores information relevant

       //{'M','D','C','L','X', 'V','I'};

       System.out.println("Going into rule 3");

       //do
       //{

       if (!VLDcheck)
       {
       for (indexCount=0; indexCount<posValues.length; indexCount++)
       {
          System.out.println("value of indexCount: " + indexCount);
          System.out.println(posValues.length);

       if (posValues[indexCount][1]==5)
       {
          countV++;
          System.out.println("found a V");
       }

       if (posValues[indexCount][1]==50)
       {
          countL++;
       }

       if (posValues[indexCount][1]==500)
       {
          countD++;
       }
       }
       }

       VLDcheck=true;
       //indexCount++;

      // }while(indexCount<posValues.length-1);


          //Rule 1: When certain numerals are repeated, the number represented by them is their sum. For
example, II = 1 + 1 = 2, or XX = 10 + 10 = 20, or, XXX = 10 + 10 + 10 = 30.
```

```
            if (posValues[m][1]==posValues[m-1][1])   // this is comparing decimal number for numeral against
previous one
            {

             // Rule 2: It is to be noted that no Roman numerals can come together more than 3 times. For example,
we cannot write 40 as XXXX
            // This needs to work from ZeroIndex not m

             System.out.println("starting rule 1");
             //if (m!=(posValues.length-2) && posValues.length>3)   // what does this rule mean, first part

             if (posValues.length>2)


            // m is current position in the numerals array

               {

               if (m!=posValues.length)  // can not check for three in row if last element in array
               {


               System.out.println("start of rule2");
               System.out.println("*********************************************************");

               if (temp1+2<=posValues.length)
               {
                  System.out.println("ffffffff");
                  System.out.println(temp1);

                  // checking for three in row same

                  if (temp1+3<=posValues.length)
                  {

                  if (posValues[temp1][1]==posValues[temp1+1][1] &&
posValues[temp1+1][1]==posValues[temp1+2][1])
               {
                  System.out.println("Three in row are the same");
                  threeInRow = (posValues[temp1][1] * 3) +threeInRow;

                  System.out.println(threeInRow);

                  processedConversion[temp1]=1;
                  processedConversion[temp1+2]=1;
                  processedConversion[temp1+1]=1;
                  temp1 = temp1+3;

               }
               }


               if (temp1+2>posValues.length &&   (posValues.length-4)>temp1)
               {
               if (posValues[temp1][1]==posValues[temp1+1][1] &&
posValues[temp1+1][1]==posValues[temp1+2][1] && posValues[temp1+2][1]==posValues[temp1+3][1])
               {
                  System.out.println("There are more than three consecutive same characters: " +
acceptedNumerals[posValues[m][0]]);
                  System.out.println("This is invalid roman numeral");
                  illegalThreeInARowState = true;
                  break;
                  //System.exit(0);
```

```
    }
    }


    }

    }
}

if (!illegalThreeInARowState)
{
    System.out.println("Amit Amlani");

    if (processedConversion[m]!=1 && processedConversion[m-1]!=1)
    {
    rule2total=posValues[m][1]+posValues[m-1][1];
      rule2=rule2+rule2total;    // they will be totalled if same
    System.out.println("So far sum:" + rule2);
    rule2State=true;
    }



     processedConversion[m]=1;
     processedConversion[m-1]=1;

    valueConsecutiveOccurences[m] = posValues[m][1];  //storing decimal value in array.

    consecutiveOccurrences=consecutiveOccurrences+2;

    // valueConsecutiveOccurences[m] this keeps a track of the roman numeral that has been repeating
    //num++;

    // this checks if there has already been consecutive occurence of the numeral adjacently
    //[m][0]=numeral position     //[m][1]=decimal

    if (m!=(posValues.length-1))
    {

    if (valueConsecutiveOccurences[m]==posValues[m+1][1] && processedConversion[m+1]!=1)
    {

       System.out.println("does this happen45");
       rule2=rule2-(rule2total/2);
       //System.out.println("sum so far: " + sum);
       rule2total=0;
       processedConversion[m+1]=1;

    }
}


// NEED TO VERIFY THIS
  if (illegalThreeInARowState)
{
    rule2=0;
}

}




}
```

```
            //Rule 5: When a Roman numeral is placed after another Roman numeral of greater value,
            //the result is the sum of the numerals. For example, VIII = 5 + 1 + 1 + 1 = 8, or,
            //XV = 10 + 5 = 15,
            // This appears to be where the code has determined if numerals are in some decent order

            System.out.println("What is value of zeroindexcount: " + zeroIndexCount);

            if (posValues[m][1]<posValues[m-1][1])
            {
               System.out.println("How many times does it enter in this section");

               System.out.println("What is m: " + posValues[m][1]);
               System.out.println("What is m-1: " + posValues[m-1][1]);


               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])][1]]);

               //*********************Need to be careful here of making it already set to 1
               //

               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])-1][1]]);

               if (processedConversion[m-1]!=1)
               {
               rule5total = posValues[m][1] + posValues[m-1][1];
               rule5 = rule5total + rule5;
               System.out.println("*******RULE5******: " + rule5);
               rule5total = 0;
               rule5State=true;
               processedConversion[m]=1;

               //UNSURE
               processedConversion[m-1]=1;


               if (posValues.length==2)
               {
                  processedConversion[m-1]=1;
               }

               if (m!=1)   // if not the first numeral
               {
                  System.out.println("ARE WER HERE******");
                  System.out.println(m);


               if (rule4Single!=0)
               {
                  System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!");
                  //rule5=rule4Single + posValues[m][1];
                  rule5=posValues[m][1] + posValues[m-1][1];
               }


            }

            }
            }
            System.out.println("at rule 6: " + total1);

           //Rule 6: When a Roman numeral is placed before another Roman numeral of greater value,
           //the result is the difference between the numerals. For example, IV = 5 - 1 = 4, or, XL = 50 - 10 = 40,
```

```
//or XC = 100 - 10 = 90

System.out.println("Rule 6");
System.out.println("Must WORK!!!!");

System.out.println(zeroIndexCount);
System.out.println(posValues[zeroIndexCount][1]);

if (posValues[zeroIndexCount][1]<posValues[zeroIndexCount+1][1])
{
   if (posValues[zeroIndexCount][1]==1 && posValues[zeroIndexCount+1][1]>10)
   {
      System.out.println("Roman Numeral I can not precede: " + posValues[zeroIndexCount+1][1]);
      IplacedIncorrect=true;
   }

   if (posValues[zeroIndexCount][1]==10 && posValues[zeroIndexCount+1][1]>100)
   {
      System.out.println("Roman Numeral X can not precede: " + posValues[zeroIndexCount+1][1]);
      IplacedIncorrect=true;
   }

   if (posValues[zeroIndexCount][1]==50 && posValues[zeroIndexCount+1][1]>=50)
   {
      System.out.println("Roman Numeral L can not precede: " + posValues[zeroIndexCount+1][1]);

      IplacedIncorrect=true;
   }

   if (posValues[zeroIndexCount][1]==5 &&
posValues[zeroIndexCount+1][1]>posValues[zeroIndexCount][1])
   {
      System.out.println("Roman Numeral V can not precede: " + posValues[zeroIndexCount+1][1]);

      IplacedIncorrect=true;
   }


   if (posValues[zeroIndexCount][1]!=5 && posValues[zeroIndexCount][1]!=50 &&
posValues[zeroIndexCount][1]!=500 && !IplacedIncorrect)

   {


   processedConversion[zeroIndexCount]=1;

   rule6total = posValues[zeroIndexCount+1][1] - posValues[zeroIndexCount][1];
   rule6 = rule6total + rule6;
   rule6total = 0;
   rule6State=true;
}

else
{  System.out.println("Illegal subtractive notation found with V, D or L");
   rule5=0;
   illegalSubtractive=true;
}
}

// Rule 7: When a Roman numeral of a smaller value is placed between two numerals of greater value,
//it is subtracted from the numeral on its right. For example, XIV = 10 + (5 - 1) = 14,
//or, XIX = 10 + (10 - 1) = 19

System.out.println("Rule 7");
 // rule 7 will not stall by rule 4  since the the right hand side character in rule 4 is always bigger.
```

```
            if (m!=posValues.length && posValues.length>2) // a numeral can not be in middle if it is the last in the
numeral array
            // it also can not be in middle if the first numeral. Hence using m index notation.

            {
               if (processedConversion[m]!=1)
               {

               //START FROM HERE AND TEST  XI

            if (m!=posValues.length-1)
            {

            if (posValues[m][1]<posValues[m-1][1] && posValues[m][1]<posValues[m+1][1])
            {

               //System.out.println("what " + (m-1));

               processedConversion[m]=1;
               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])][1]]);
               processedConversion[m+1]=1;
               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m+1])][1]]);

               rule7total = posValues[m+1][1] - posValues[m][1];
               rule7 = rule7total + rule7;
               rule7total = 0;

               System.out.println("total at rule 7:" + rule7);

               }

            }

            }
         }
            zeroIndexCount++;
            temp1++;

            System.out.println("This is overall total:" + (total1));
      }
   }


      // THIS WILL NOW CHECK IF EACH NUMBER HAS OR HAS NOT BEEN PROCESSED IN CHECKING SO far

      int counter=0;
      int unTotal=0;

      for (int c: processedConversion)
      {
         System.out.println("This is value of c: " + c);
         System.out.println("This is the numeral value: " + posValues[counter][1]);

         if (c!=1)
         {
            unTotal = posValues[counter][1] + unTotal;
            System.out.println("This has not been added to the total" + unTotal);
            System.out.println("This is the position: " + counter);
         }

         counter++;
      }
```

```
    // this is part of rule 3

    if (countD>1 || countL>1 || countV>1)
    {
       System.out.println("Invalid roman numeral. Numeral V or D or L has occured more than once");
       rule3Fail = true;
    }

     if (inputtedNumerals.length==2)  // checking number with 2 numerals
 {
    if (posValues[0][0]==posValues[1][0] && rule3Fail==false)  // if values identical straight forward conversion
    {
       twoNumerals = posValues[0][1] + posValues[1][1];


    System.out.println("Conversion is double:" + (twoNumerals));

    //System.out.println("XL: " +  posValues[0][1]);
    processedConversion[0]=1;

    processedConversion[1]=1;
    //System.out.println("This is value marked with 1: " +  posValues[1][1]);


    }
 }

 System.out.println("number of Vs: " + countV);
 System.out.println("number of Ls: " + countL);
 System.out.println("number of Ds: " + countD);

 System.out.println("rule4: " + rule4);
 System.out.println("rule2: " + rule2);
 System.out.println("rule4Single: " + rule4Single);
 System.out.println("rule5: " + rule5);
 System.out.println("rule6: " + rule6);
 System.out.println("rule7: " + rule7);
 System.out.println("threeinarow:" + threeInRow);
 System.out.println("Untotal:" + unTotal);

if (rule3Fail)
    {
       System.out.println("More than 2 instances of V, L and D:");
       System.out.println("D: " + countD);
       System.out.println("V: " + countV);
       System.out.println("L: " + countL);
       System.exit(0);
    }


if (!illegalSubtractive && !IplacedIncorrect && !illegalThreeInARowState && !rule3Fail)
 {
    System.out.println(numeralsToString + "is a VALID roman numeral");

     if (unTotal!=0 && rule6!=0 && rule5!=0 && rule4Single!=0)
     {
        System.out.println("total1: " + (rule4Single + unTotal + rule5));
        System.exit(0);
     }


    if (rule4Single!=0 && unTotal!=0)
    {
       System.out.println("total2: " + (rule4Single + unTotal));
```

```
    System.exit(0);
   }

   if (rule4Single!=0 && rule2!=0 && rule5!=0)
   {
    System.out.println("total3: " + (rule4Single + rule2 + rule5));
    System.exit(0);
   }

   if (rule4Single==0 && rule2!=0 && rule5!=0 && rule4!=0)
   {
    System.out.println("total4: " + (rule4 + rule2 + rule5));
    System.exit(0);
   }

   if (unTotal==0 && rule5==0 && rule2==0 && rule4==0 && threeInRow!=0)
   {
    System.out.println("Total5: " + (threeInRow));
   System.exit(0);
   }

   if (unTotal!=0 && rule5!=0 && rule6!=0)
   {
   System.out.println("Total6: " + (unTotal+rule6));
   System.exit(0);
   }


    if (unTotal!=0 && rule5!=0)
   {
   System.out.println("Total6: " + (unTotal+rule5));
   System.exit(0);
   }

   if (unTotal!=0 && rule5==0 && rule2==0 && rule4==0 && threeInRow!=0)
   {
    System.out.println("Total7: " + (unTotal+threeInRow));
   System.exit(0);
   }

   if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0 && threeInRow!=0)
   {
    System.out.println("Total8: " + (threeInRow+rule2));
    System.exit(0);
   }

   if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0 && unTotal!=0)
   {
    System.out.println("Total9: " + (rule2+unTotal));
    System.exit(0);
   }

   if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0)
   {
    System.out.println("Total10: " + rule2);
    System.exit(0);
   }

   if (rule4Single!=0 && rule5!=0 && rule6!=0)
   {
    System.out.println("Total11: " + (rule4Single+rule5));
    System.exit(0);
   }

if (unTotal!=0 && rule5==0 && rule2==0 && rule4==0 && rule4Single==0)
```

```
        {
           System.out.println("Total12: " + (unTotal));
        System.exit(0);
        }

      if (unTotal!=0 && rule4!=0)
        {
        System.out.println("Total13: " + (unTotal + rule4));
        System.exit(0);
        }

       if (rule5!=0 && rule6!=0 && unTotal!=0)
        {
           System.out.println(numeralsToString + "is an invalid roman numeral");
           System.out.println(numeralsToString + "INVESTIGATE CODE OR CONDITIONAL IF LOOPS FOR TOTAL IF
ENTER OCCURS");

           System.exit(0);
        }

       if (rule6!=0 && rule5!=0)
        {
        System.out.println("Total14: " + (rule6+rule5));
        System.exit(0);
        }



      if (rule6!=0 && rule2!=0)
      {
         System.out.println("Total 15: " + (rule6+rule2));
         System.exit(0);
      }

        if (rule5!=0 && rule2!=0 && threeInRow==0 && rule4Single!=0)
      {
         System.out.println("Total16: " + (rule5+rule2));
         System.exit(0);
      }


      if (rule5!=0 && rule2!=0 && threeInRow!=0)
      {
         System.out.println("Total17: " + (rule5+rule2));
         System.exit(0);
      }


       if (rule5!=0 && rule2!=0)
      {
         System.out.println("Total18: " + (rule5+rule2));
         System.exit(0);
      }



      if (rule5!=0 && unTotal!=0)
        {
           System.out.println("Total19: " + (rule5+unTotal));
           System.exit(0);
        }

          if (rule5!=0 && rule4!=0)
        {
```

```
      System.out.println("Total20: " + (rule5 + rule4));
      System.exit(0);
    }

    if (rule5!=0 && threeInRow!=0 && rule4Single!=0)
    {
      System.out.println("Total21: " + (rule5+threeInRow));
      System.exit(0);
    }


    if (rule5!=0 && threeInRow!=0)
    {
      System.out.println("Total22: " + (rule5+threeInRow));
      System.exit(0);
    }

    if (rule5!=0)
    {
      System.out.println("Total23: " + (rule5));
      System.exit(0);
    }

    if (rule2!=0 && unTotal!=0 && rule4Single!=0)
    {
      System.out.println("Total24: " + (rule2 + unTotal + rule4Single));
      System.exit(0);

    }


    if (rule2!=0 && rule4Single!=0)
    {
      System.out.println("Total25: " + (rule2 + rule4Single));
      System.exit(0);
    }

   if (rule2!=0)
    {
      System.out.println("Total26: " + (twoNumerals));
      System.exit(0);
    }


    if (rule4Single!=0)
    {
      System.out.println("Total27 is: " + (unTotal + rule4Single));
      System.exit(0);
    }

      if (rule4!=0)
    {
      System.out.println("Total28 is: " + (rule4));
      System.exit(0);
    }
 }
    System.out.println(numeralsToString + "is NOT a VALID roman numeral");
    System.out.println("Address the issues outputted for a valid Roman numeral");


 //System.out.println("this is the overall sum: " + (rule2+rule4Single+rule5+rule6+rule7));
// System.out.println("this is the overall sum: " + (rule5 + rule2));

 // This is the best logic found on the internet for roman numeral conversion.
 // Logic will be coded one by one and see if it works overall
```

```
/*

   It is necessary for us to remember the rules for reading and writing Roman numbers in order to avoid mistakes. Here is a list of the basic rules for Roman numerals.

   Rule 1: When certain numerals are repeated, the number represented by them is their sum. For example, II = 1 + 1 = 2, or XX = 10 + 10 = 20, or, XXX = 10 + 10 + 10 = 30.
   Rule 2: It is to be noted that no Roman numerals can come together more than 3 times. For example, we cannot write 40 as XXXX
   Rule 3: The xcletters V, L, and D are not repeated.
   Rule 4: Only I, X, and C can be used as subtractive numerals. There can be 6 combinations when we subtract. These are IV = 5 - 1 = 4; IX = 10 - 1 = 9; XL = 50 - 10 = 40; XC = 100 - 10 = 90; CD = 500 - 100 = 400; and CM = 1000 - 100 = 900
   Rule 5: When a Roman numeral is placed after another Roman numeral of greater value, the result is the sum of the numerals. For example, VIII = 5 + 1 + 1 + 1 = 8, or, XV = 10 + 5 = 15,
   Rule 6: When a Roman numeral is placed before another Roman numeral of greater value, the result is the difference between the numerals. For example, IV = 5 - 1 = 4, or, XL = 50 - 10 = 40, or XC = 100 - 10 = 90
   Rule 7: When a Roman numeral of a smaller value is placed between two numerals of greater value, it is subtracted from the numeral on its right. For example, XIV = 10 + (5 - 1) = 14, or, XIX = 10 + (10 - 1) = 19
   Rule 8: To multiply a number by a factor of 1000 a bar is placed over it.
   Rule 9: Roman numerals do not follow any place value system.
   Rule 10: There is no Roman numeral for zero (0).
*/

   }

}
```

FEW OUTPUTS

Scenario 1: Trying a really long tricky conversion: MMCDLXIX

```
Welcome to Online IDE!! Happy Coding :)
Enter number numerals in the number to be converted to decimal:

8
Enter roman numeral 1 of 8:
M
Enter roman numeral 2 of 8:
M
Enter roman numeral 3 of 8:
C
Enter roman numeral 4 of 8:
D
Enter roman numeral 5 of 8:
L
Enter roman numeral 6 of 8:
X
Enter roman numeral 7 of 8:
I
Enter roman numeral 8 of 8:
X
This will be converted to decimal:MMCDLXIX
```

```
number of Vs: 0
number of Ls: 1
number of Ds: 1
rule4: 409
rule2: 2000
rule4Single: 0
rule5: 60
rule6: 400
rule7: 400
threeinarow:0
Untotal:0
MMCDLXIX is a VALID roman numeral
total4: 2469
```

THE PROGRAM HAS 28 DIFFERENT TOTAL SCENARIOS

Scenario 2: Trying a shorter conversion: XVL

```
X
Enter roman numeral 2 of 3:
V
Enter roman numeral 3 of 3:
L
This will be converted to decimal:XVL
```

```
Roman Numeral V can not precede: 50
Illegal subtractive notation found with V, D or L
```

```
number of Vs: 1
number of Ls: 1
number of Ds: 0
rule4: 0
rule2: 0
rule4Single: 0
rule5: 0
rule6: 0
rule7: 0
threeinarow:0
Untotal:50
XVL is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

These are now some sequences which will trigger errors. I used this a foundation to put any illegal roman numerals aside:

XCX

```
Enter number numerals in the number to be converted to decimal:
3
Enter roman numeral 1 of 3:
X
Enter roman numeral 2 of 3:
C
Enter roman numeral 3 of 3:
X
```

```
1. The numeral X(=10)  cannot be placed after a group of numerals written in subtractive notation: XC
Of the same range: ( = 10-90).
```

```
number of Vs: 0
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 0
rule4Single: 90
rule5: 0
rule6: 90
rule7: 0
threeinarow:0
Untotal:10
XCX is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

IVC

```
Enter roman numeral 1 of 3:

I
Enter roman numeral 2 of 3:
V
Enter roman numeral 3 of 3:
C
This will be converted to decimal:IVC
```

```
3. A group of numerals written in subtractive notation, of lower value,
IV ( = 4), cannot precede a numeral of larger value, C( = 100.
```

```
number of Vs: 1
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 0
rule4Single: 4
rule5: 0
rule6: 4
rule7: 0
threeinarow:0
Untotal:100
IVC is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

CCM

```
Enter roman numeral 1 of 3:
C
Enter roman numeral 2 of 3:
C
Enter roman numeral 3 of 3:

M
This will be converted to decimal:CCM
```

```
2. A numeral (a letter) of lower valueC(=100) cannot precede a group of numerals written in subtractive notation,
CM( =900).
```

```
CCM is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

MLCXIV

```
M
Enter roman numeral 2 of 6:
L
Enter roman numeral 3 of 6:
C
Enter roman numeral 4 of 6:
X
Enter roman numeral 5 of 6:
I
Enter roman numeral 6 of 6:
V
This will be converted to decimal:MLCXIV
```

```
Roman Numeral L can not precede: 100
Illegal subtractive notation found with V, D or L
```

```
rule4: 0
rule2: 0
rule4Single: 4
rule5: 110
rule6: 0
rule7: 0
threeinarow:0
Untotal:0
MLCXIV is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

MMMCCCXIV

Initially my total was wrong.. But I had to add another variant for total calculation. This should not affect validation of others

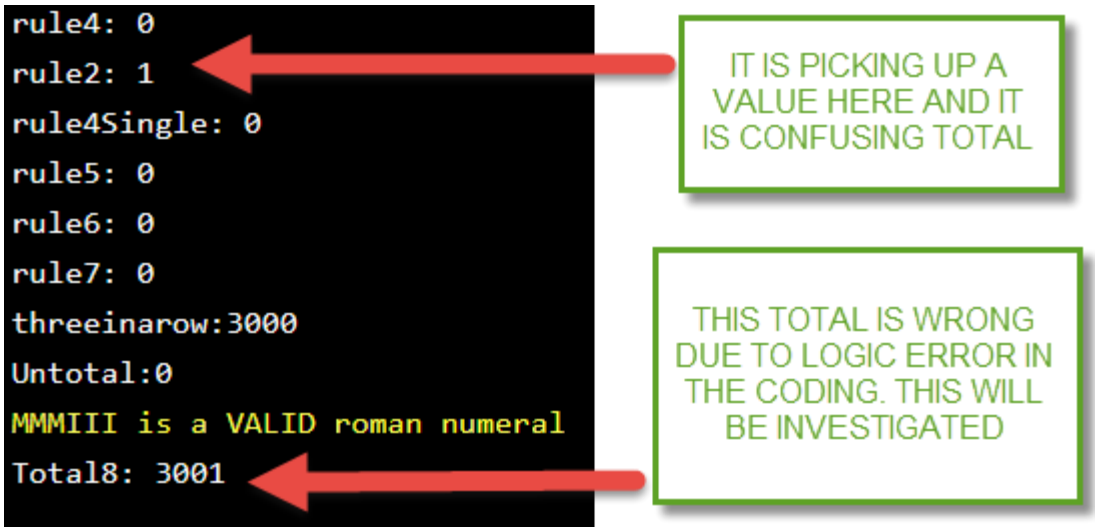
```
Enter roman numeral 1 of 9:

M
Enter roman numeral 2 of 9:
M
Enter roman numeral 3 of 9:
M
Enter roman numeral 4 of 9:

C
Enter roman numeral 5 of 9:

C
Enter roman numeral 6 of 9:
C
Enter roman numeral 7 of 9:

X
Enter roman numeral 8 of 9:
I
Enter roman numeral 9 of 9:
V
This will be converted to decimal:MMMCCCXIV
```

```
rule4: 0
rule2: 100
rule4Single: 4
rule5: 0
rule6: 0
rule7: 0
threeinarow:3000
Untotal:10
MMMCCCXIV is a VALID roman numeral
total2_temp: 3014
```

```
number of Vs: 2
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 10
rule4Single: 0
rule5: 0
rule6: 0
rule7: 0
threeinarow:3000
Untotal:1
More than 2 instances of V, L and D:
D: 0
V: 2
L: 0
MMMVVI is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

MMMIII

HOWEVER ON THIS INSTANCE, THERE IS CURRENTLY A FLAW IN THE CODING. SINCE IT IS PROCESSING THE MMM BUT NOT III



MMMVVI

```
Enter roman numeral 1 of 6:

M
Enter roman numeral 2 of 6:
M
Enter roman numeral 3 of 6:
M
Enter roman numeral 4 of 6:

V
Enter roman numeral 5 of 6:
V
Enter roman numeral 6 of 6:
I
This will be converted to decimal:MMMVVI
```

```
Invalid roman numeral. Numeral V or D or L has occured more than once
number of Vs: 2
number of Ls: 0
number of Ds: 0
```

MMMCCCXXXIII

IT ALSO FAILS THIS CONVERSION FOR SIMILAR REASON. IT CAN NOT HANDLE THREE IDENTICAL VALID NUMERALS IN A ROW REPETITIVELY. **THIS HAS NOW BEEN FIXED.**

AT THIS POINT, I HAVE CHOSEN TO GO FOR A DIFFERENT APPROACH IN CODING.
I HAVE CHOSEN TO:
* COMPLETE TOTAL FOR ALL SUBTRACTIVE NOTATIONS.
* APPLY ANY RULES TO ENSURE ILLEGAL NUMERALS ARE FLAGGED UP.
* IF NO ERRORS, THEN SIMPLY ADD THE REMAINING

NUMERALS TO THE RUNNING TOTAL FOR SUBTRACTIVE NOTATIONS.
THIS IS TO ENSURE I AM NOT CREATING SEVERAL TOTAL SCENARIOS, WITHOUT UNDERSTANDING REASON.

SOME TESTING AS BELOW, BOTH CODES ARE RUNNING IN PARALLEL. IT WILL ALLOW ME TO CHECK FOR ANY ISSUES. AS FAR AS I AM CONCERNED, NEITHER CODE IS PERFECT.

TEST SCENARIOS:

XMMC – PASS

```
This will be converted to decimal:XMMC
the length is:4
numerals in the roman numeral:4
```

```
********** GRAND TOTAL**********
XMMC is an INVALID roman numeral
Address the issues outputted for a valid Roman numeral
********************************
```

```
****************************************
OLD PROGRAMME LOGIC

****************************************
XMMC is the roman numeral
number of Vs: 0
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 2000
rule4Single: 0
rule5: 1100
rule6: 0
rule7: 0
threeinarow:0
Untotal:2110
XMMC is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

MXXXIV – 1034 (PASS)

```
********** GRAND TOTAL**********
MXXXIV is a VALID roman numeral
runningTotal:4
Not added total:1030
**TOTAL: 1034
********************************
```

```
****************************************
OLD PROGRAMME LOGIC

****************************************
MXXXIV is the roman numeral
number of Vs: 1
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 30
rule4Single: 4
rule5: 1010
rule6: 0
rule7: 0
threeinarow:0
Untotal:1030
total2: 1034
```

MMMXCIV – 3094 (FAIL - old logic).

```
This will be converted to decimal:MMMXCIV
the length is:7
```

```
********** GRAND TOTAL**********
MMMXCIV is a VALID roman numeral
runningTotal:94
Not added total:3000
**TOTAL: 3094
********************************
```

```
*****************************************
OLD PROGRAMME LOGIC

*****************************************
MMMXCIV is the roman numeral
number of Vs: 1
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 3000
rule4Single: 0
rule5: 0
rule6: 0
rule7: 0
threeinarow:0
Untotal:3000
Total9: 3000
```

THIS IS NOW FOCUSSING ON INCORRECT SYNTAX ROMAN NUMERALS:

MIVCXVI - PASS

```
This will be converted to decimal:MIVCXVI
the length is:7
```

```
3ver1. A group of numerals written in subtractive notation, of lower value,
IV ( = 4), cannot precede a numeral of larger value, C( = 100.
```

```
Roman Numeral V can not precede: 100
Illegal subtractive notation found with V, D or L
```

```
******************** GRAND TOTAL*************
MIVCXVI is an INVALID roman numeral
Address the issues outputted for a valid Roman numeral
*********************************
```

```
OLD PROGRAMME LOGIC

*****************************************
MIVCXVI is the roman numeral
number of Vs: 2
number of Ls: 0
number of Ds: 0
rule4: 0
rule2: 0
rule4Single: 4
rule5: 6
rule6: 4
rule7: 0
threeinarow:0
Untotal:1116
More than 2 instances of V, L and D:
D: 0
V: 2
L: 0
MIVCXVI is NOT a VALID roman numeral
Address the issues outputted for a valid Roman numeral
```

THIS IS NOW TESTING A LONG EXTENDED NUMERAL AS PREVIOUSLY TO ENSURE NO LOGIC ERRORS HAVE APPEARED:

MMCDLXIX = 2469 (FAIL – OLD LOGIC)
It passed previously, but since lots of general code has changed, it is now not passing.
No issues new code:

```
********************* GRAND TOTAL*************
MMCDLXIX is a VALID roman numeral
runningTotal:409
Not added total:2060
**TOTAL: 2469
*********************************
```

```
*****************************************
OLD PROGRAMME LOGIC

*****************************************
MMCDLXIX is the roman numeral
number of Vs: 0
number of Ls: 1
number of Ds: 1
rule4: 0
rule2: 2000
rule4Single: 0
rule5: 60
rule6: 400
rule7: 0
threeinarow:0
Untotal:2060
Total6: 2460
```

INCORRECT

DLXIV – 564 (PASS)

```
******************** GRAND TOTAL************
DLXIV is a VALID roman numeral
runningTotal:4
Not added total:560
**TOTAL: 564
********************************
```

```
*****************************************
OLD PROGRAMME LOGIC

*****************************************
DLXIV is the roman numeral
number of Vs: 1
number of Ls: 1
number of Ds: 1
rule4: 0
rule2: 0
rule4Single: 4
rule5: 60
rule6: 0
rule7: 0
threeinarow:0
Untotal:560
total2: 564
```

FINAL CODE: IT CONTAINS LOTS OF COMMENTS WHICH MIGHT NOT NECESSARILY BE IN CORRECT LOCATION DUE TO CHANGING CODE, SCREEN SYSTEM.OUT.PRINTLN

```
//best
// USE THIS BUT BE EXTRMELY CAREFUL TO KEEP checking
//XCX  IVC   CCM  (these are supppose to error) - this is working fine.....
//They are critical for coding.
//Currently up to line 914
// issue currently with order of notations    CM XI  is valid   XI CM (not valid)
//but software showing both as ok - NEEDS TO BE FIXED WITH PRIORITY
//ISSUE IS ONLY WITH 2 NOTATIONS
//logic of programme alsmost complete:
// need to add total for all subtractive notations...
// then apply other rules for invalid numbers
// then the untotal will be all the other characters which have not been processed
//best
// USE THIS BUT BE EXTRMELY CAREFUL TO KEEP checking
//XCX  IVC   CCM  (these are supppose to error)
// It also does not get to the above conditions if numeral is longer such as MIVC..  now fixed
//They are critical for coding.

//NEED TO RE-CHECK LOGIC OF HOW twoSubtractiveNotations IS SET TO true - TOTAL IS DOUBLE (WRONG)

//IVCM now fails as expected


// added code to support both directions looking in rule 3
// XCX and IVC still ok... but CCM overruns i counter to 5.. bigger than length#
// trying something like IVC to reach  rule 3 ver2....  it reaches max index

// OUTSTANDING.
//CMIV - it is doubling value to 1808 instead of 909...
// this is issue of CM (i index)   IV (j index)  comparing to   CM (j index) and IV (i index)

//Everything else is fixedg




import java.util.Scanner;
import java.util.Arrays;
/*
Online Java - IDE, Code Editor, Compiler
valu`va
Online Java is a quick and easy tool that helps you to build, compile, test your programs online.
*/
// need to address finding match of 4 consecutive numerals
// need to address errors on word document
// can not handle VII  or VIII
//switching technique


public class Main
{
    public static void main(String[] args) {

     System.out.println("Welcome to Online IDE!! Happy Coding :)");

     char[] acceptedNumerals = new char[]{'M','D','C','L','X', 'V','I'};
     //These are all accepted roman numerals
```

```
      int number; //This is set once end user specifies number numerals in their number to be inputted

      boolean noMatch=true;  // this flag is used to check if the numeral entered is a valid one

      int j;  //used to increment the character array containing user input of numeral
      int count=0;  // this keeps track number times end user has entered a character. It can be decremented if it
fails validation.

      Scanner reader=null;

      // This is used to ascertain how many digits end user wants
      reader = new Scanner(System.in); // Reading from System.in
      System.out.println("Enter number numerals in the number to be converted to decimal:");
      number=reader.nextInt();

      char inputtedNumerals[] = new char[number];   // this creates character array with end user specified size
      int [][] posValues = new int [number][2];  // this is used to keep track of:

       // this is 2D array. 'number' simply increments with array index of the roman numeral in character array
      // the multidimensal aspect at 0 index will store:  index of which numeral i.e  M is 0, D is 1
                        //aspect at 1 index will store:  decimal conversion
      // This will support later analysis.

      //all values will be set to 1 to ensure a repeat conversion is not processed;

do
{
reader = new Scanner(System.in); // Reading from System.in
System.out.println("Enter roman numeral " + (count+1) + " of " + number + ":");   // end user prompted to enter
roman numeral
inputtedNumerals[count]=reader.next().charAt(0);
count++;

for (j=0; j<inputtedNumerals.length;j++)  // this will check each item in the inputted area against 7 roman
numerals.
                              // the condition could equally be set to count for better loop efficiency if the
                              //number entered is long
{

   for (int k=0; k<acceptedNumerals.length;k++)  // this goes through roman numerals
   {

   if (inputtedNumerals[j]==acceptedNumerals[k])  // this checks to see if the roman numerals entered are valid
   {

      posValues[count-1][0]=k;        // Variable k keeps a record of index of the roman numeral stored in
acceptedNumerals

      // This switch statement now checks 'k'validposition in hierarcy of the 7 numerals.
      //'M','D','C','L','X', 'V','I'

      // Depending on the roman numeral value, it will also store the decimal conversion into posValues array.
      switch(k)
      {
         case 0:
            posValues[count-1][1]=1000;
            break;

         case 1:
            posValues[count-1][1]=500;
            break;

         case 2:
            posValues[count-1][1]=100;
```

```
            break;

         case 3:
            posValues[count-1][1]=50;
            break;

         case 4:
            posValues[count-1][1]=10;
            break;

         case 5:
            posValues[count-1][1]=5;
            break;

         case 6:
            posValues[count-1][1]=1;
            break;
      }


      noMatch=false;   // This sets flag stating match has been found.
      break;

      }

}
// This will now give information if the user input is valid for each numeral.
// If found to be incorrect, the count will go back and end user will need to enter it again.

 if (noMatch==true)
   {
     System.out.println("Please input a successful roman numeral:  M  D  C  L  X  V  I");
     count=count-1;
     break;
   }

}

} while (count<inputtedNumerals.length);  // do while loop will end once count reaches the input user for length
of roman numerals.


String str = String.valueOf(inputtedNumerals);
System.out.println("This will be converted to decimal:" + str);  // This informs end user of the roman numerals to
be converted

performConversion pc = new performConversion(acceptedNumerals, inputtedNumerals,posValues,str); // calls
constructor in class to perform conversion
}
}

class performConversion
{
   int tempA=0;
   int tempB=0;
   boolean FlagSet=false;

   int sumNotAdded=0;
   int rule4;
   boolean rule3Fail=false;
   char[] inputtedNumerals;  // this contains numerals entered by end user

   char[] acceptedNumerals;  // this contains array of all accepted numerals
   int [][] posValues; // this keeps track for the  index for numeral and also its conversion into decimal
   int total;
```

```
   int sum;
   int consecutiveOccurrences;
   String numeralsToString;
   boolean correctOrder=false;

   int oneNumeral;
   int twoNumerals;
   boolean illegalSubtractive=false;
   boolean illegalFourInARowState=false;
   int temp1=0;
   boolean valueBeforeSubtractiveNotation=false;
   int runningTotal=0;
   boolean validNumber=false;

    // This is the constructor used to call the converter method.

   public performConversion (char[] aNumerals, char [] inputNumerals, int [][] pValues, String numeralsToString)
   {

      int pc[] = new int[pValues.length];
      this.acceptedNumerals=aNumerals;
      this.inputtedNumerals=inputNumerals;
      this.posValues=pValues;
      this.numeralsToString=numeralsToString;

      converter(pc);   // this variable has been passed into the method since for some reason it did not pick up the
variable even though scope was correct.
   }
   public void converter(int [] processedConversion)
   {
      // There will be NO issues in performing conversion of a single roman numeral
      System.out.println("the length is:" + inputtedNumerals.length);
      if (inputtedNumerals.length<2)    // This functions correctly.
      {
            System.out.println("Conversion is single:" + posValues[0][1]);


      }

          // ONCE ALL PRINCIPLES ARE UNDERSTOOD FOR A VALID ROMAN NUMERAL, THE CODE WILL BE
IMPLEMENTED FURTHER


      System.out.println("numerals in the roman numeral:" + posValues.length);

      int valueConsecutiveOccurences[] = new int [posValues.length];
      int indexCount=0;

      // These values can only occur once in roman numeral. A counter is kept.
      int countV=0;
      int countL=0;
      int countD=0;

      int zeroIndexCount=0;  // on instances, the counter has to start from zeroindex notation

      int rule4Single=0;  //this keeps a total of rule 4

      int rule5=0;  // this keeps total at rule 5
      int rule5total;
      boolean rule5State=false;  // state is set if rule 5 is entered

      int rule6=0;     // this keeps total at rule 6
      int rule6total;
      boolean rule6State=false;  // state is set if rule 6 is entered
```

```
int rule7=0;    // this keeps total at rule 7
int rule7total;
boolean rule7State=false;  // state is set if rule 7 is entered

// used for

 //Rule 4: Only I, X, and C can be used as subtractive numerals.
   //There can be 6 combinations when we subtract.
   //These are IV = 5 - 1 = 4; IX = 10 - 1 = 9; XL = 50 - 10 = 40;
   // XC = 100 - 10 = 90; CD = 500 - 100 = 400; and CM = 1000 - 100 = 900

   //Another useful IMPORTANT information found on a site is that two subtractive numerals can not be
adjacent such as:
   // CDCM since CM is higher however   CM CD is also invalid (range is 100-900)

   // The following is invalid:   IXIV even though IV is smaller than IX. This is because they are both in same
range 1-9
   // In that respect,   XC XL is invalid..  both notations are  10-90
   // XCIV is ok

   //{'M','D','C','L','X', 'V','I'};

   // I have partially implemented the rule here since it is too complex.

String IV =  "IV";
String IX =  "IX";
String XL =  "XL";
String XC =  "XC";
String CD=   "CD";
String CM=   "CM";

int threeInRow=0;

String [] subtractiveNumerals =  new String []{IV,IX,XL,XC,CD,CM};

int [][] subtractiveNumeralsRange = new int[6][2];

subtractiveNumeralsRange[0][0] = 1;   subtractiveNumeralsRange[0][1] = 4;
subtractiveNumeralsRange[1][0] = 1;   subtractiveNumeralsRange[1][1] = 9;
subtractiveNumeralsRange[2][0] = 10;   subtractiveNumeralsRange[2][1] = 40;
subtractiveNumeralsRange[3][0] = 10;   subtractiveNumeralsRange[3][1] = 90;
subtractiveNumeralsRange[4][0] = 100;   subtractiveNumeralsRange[4][1] = 500;
subtractiveNumeralsRange[5][0] = 100;   subtractiveNumeralsRange[5][1] = 900;


int sum1=0;
int total1=0;
boolean flagError=false;

boolean result=false;
int total2=0;


boolean twoSubtractiveNotations=false;
boolean singleSubtractiveNotations=false;
boolean IplacedIncorrect=false;



boolean rule2State=false;
int tempBefore=0;

int rule2=0;
int rule2total=0;
boolean invalidSubtractiveNumeral=false;
```

```
      int temp=0;

      boolean VLDcheck=false;

      boolean illegalThreeInARowState=false;
      int rangeLower=0;
      int rangeUpper=0;

      String checkComplete[] = new String[subtractiveNumerals.length];

      //----------------------------------------

      // This has been completed outside main loop because it is processing a string to check values
      // this needs to go inside main loop. otherwise can not check the index of posValues

      System.out.println("In rule 4");
      for (int i=0; i<subtractiveNumerals.length; i++)   // this is going through all 6 combos IV, IX, XL.....
      {
         System.out.println("**********************");
         System.out.println("Which lop number i:" + i);

         for (int j=0; j<subtractiveNumerals.length; j++ )   // this is going through all 6 combos IV, IX, XL.....
         {
            System.out.println("Which lop number j:" + j);
            // NEED MORE LOGIC TO PREVENT IT FROM COMPARING AGAINST ITSELF.
            // OTHERWISE IT WILL PREVENT SINGLE OCCURENCE OF SUBTRACTIVE NOTATION FROM BEING
PROCESSED

            if (j==i)
            {
               continue;
            }

            // this ensures that there are matches against two subtractive notations before processing
            if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 &&
numeralsToString.indexOf(subtractiveNumerals[j])!=-1)
            {
               FlagSet=true;
                twoSubtractiveNotations = true;
                System.out.println("TWO SUBTRACTIVE NOTATIONS FOUND:");
                System.out.println("**************************");

               // this checks if subtractive notation lower value appears at index before higher subtractive notation

               if (numeralsToString.indexOf(subtractiveNumerals[i]) <
numeralsToString.indexOf(subtractiveNumerals[j]))

               {
                  checkComplete[i]=subtractiveNumerals[i]+subtractiveNumerals[j];
                  if(checkComplete[i].indexOf(subtractiveNumerals[j])==-1)
                  {

                  // THERE IS ISSUE IN THIS PART OF CODE:
                  // FOR INSTANCE IF IT DETECTS   M CM IV    It will first find circumstance ok since  IV is in i loop and
CM is in j loop
                  //and hence since index of IV will appear higher, it will decide not to proceed with this.
                  // HOWEVER as the i loop progresses  it will become CM in i loop.. It will then find  IV in the j loop.
                  // In this instance CM will appear lower in the index notation... hence the loop will validate as true
and proceed.
                  // Need a way to stop this from occuring.


                  System.out.println(numeralsToString + " is invalid");
                  System.out.println("Incorrect roman numeral. " +  subtractiveNumerals[i] + " portion should be
before: " + subtractiveNumerals[j]);
```

```
                System.out.println("A group of numerals written in subtractive notation, of lower value, " +
subtractiveNumerals[i] + " ( =" + subtractiveNumeralsRange[i][1]+"), cannot precede another group of numerals
written in subtractive notation," + subtractiveNumerals[j] + " ( =" + subtractiveNumeralsRange[j][1]+").");

                illegalSubtractive=true;
                 //
                 invalidSubtractiveNumeral=true;

              }

              }


            // this now checks the subtractive notations in correct order

            if (numeralsToString.indexOf(subtractiveNumerals[i]) >
numeralsToString.indexOf(subtractiveNumerals[j]))
            {
              //Both these subtractive numerals appear. a case of now checking if range is same

            System.out.println(subtractiveNumerals[i]);
            System.out.println("This should print out start of the range:" + subtractiveNumeralsRange[i][0]);
            System.out.println("This should print out end of the range:" + subtractiveNumeralsRange[i][1]);

            System.out.println(subtractiveNumerals[j]);
            System.out.println("This should print out start of the range:" + subtractiveNumeralsRange[j][0]);
            System.out.println("This should print out end of the range:" + subtractiveNumeralsRange[j][1]);

              if (subtractiveNumeralsRange[i][0] == subtractiveNumeralsRange[j][0])  // these subtractive notations
are in the same range
              {
                System.out.println(subtractiveNumerals[i] + " is in the range: " + subtractiveNumeralsRange[i][0] + "
- " + subtractiveNumeralsRange[i][1]);
                System.out.println(subtractiveNumerals[j] + " is in the range: " + subtractiveNumeralsRange[j][0] + "
- " + subtractiveNumeralsRange[j][1]);
                System.out.println("They are both in the same range");


                processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
                temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
                processedConversion[temp]=1;

                System.out.println("REACH HERE!!!!!!!!!!");
                processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]=1;
                temp = numeralsToString.indexOf(subtractiveNumerals[j]) + 1;
                processedConversion[temp]=1;


                rule4=0;

              }

              if (subtractiveNumeralsRange[i][0] != subtractiveNumeralsRange[j][0])  // this ensures that subtractive
notations are not in the same range
              {

                if(processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]!=1 &&
processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]!=1 )
                {

              if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 &&
numeralsToString.indexOf(subtractiveNumerals[j])!=-1)
              {
                System.out.println("correct order");
                correctOrder=true;
```

```
            System.out.println(subtractiveNumerals[i] + "   " +
numeralsToString.indexOf(subtractiveNumerals[i]));
        System.out.println(subtractiveNumerals[j] + "   " + numeralsToString.indexOf(subtractiveNumerals[j]));

            //CAREFUL THESE BEEN COMMENTED
            processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
            temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
            processedConversion[temp]=1;

            processedConversion[numeralsToString.indexOf(subtractiveNumerals[j])]=1;
            temp = numeralsToString.indexOf(subtractiveNumerals[j]) + 1;
            processedConversion[temp]=1;


            switch (subtractiveNumerals[i])
            {
               case "IV":
                  runningTotal = runningTotal + 4;
                  break;

               case "IX":
                  runningTotal = runningTotal + 9;
                  break;

               case "XL":
                  runningTotal = runningTotal + 40;
                  break;

               case "XC":
                  runningTotal = runningTotal + 90;
                  break;

               case "CD":
                  runningTotal = runningTotal + 400;
                  break;

               case "CM":
                  runningTotal = runningTotal + 900;
                  break;

               default:
               invalidSubtractiveNumeral = true;
            }

            switch (subtractiveNumerals[j])
            {
               case "IV":
                  runningTotal = runningTotal + 4;
                  break;

               case "IX":
                  runningTotal = runningTotal + 9;
                  break;

               case "XL":
                  runningTotal = runningTotal + 40;
                  break;

               case "XC":
                  runningTotal = runningTotal + 90;
                  break;

               case "CD":
                  runningTotal = runningTotal + 400;
```

```
                    break;

                case "CM":
                    runningTotal = runningTotal + 900;
                    break;

                default:
                invalidSubtractiveNumeral = true;
              }

            }
            }
          }

          }
          }
        }


        // this confirms if a match is found.  -1 value is no match. 0 is a match.
        System.out.println(subtractiveNumerals[i] + "   " + numeralsToString.indexOf(subtractiveNumerals[i]));
        boolean correctOrder=false;
        System.out.println("total after rule4:" + rule4);


        // this now is just simply going through all subtractive notations if there is no other existing
        // this can not run if the above circumstance is true.
          //System.out.println("Was not expecting this loop to enter with " +  subtractiveNumerals[i] +
subtractiveNumerals[j]);

          System.out.println(twoSubtractiveNotations);

          if (!twoSubtractiveNotations)

        {
          System.out.println("What is I:" + i);
          System.out.println("BE HERE!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1)  // This double checks that there is only one
subtractive notation
          {System.out.println("BE HERE AGAIN!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
            //if (processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]!=1)
            //{
            System.out.println("BE HERE AGAIN2!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
             // **This is now checking if X for instance is occuring in the numerals and also subtractive notations
            // ** for instance   XCX

            //the code has not referenced range at all.. this will cause code to fail in scenarios such as MCM
where it is VALID


            System.out.println("ASDASDBGDFSDGFDGFDG");
            System.out.println(subtractiveNumerals[i]);
            runningTotal = runningTotal + subtractiveNumeralsRange[i][1];

            if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1)
            // if it finds a match of subtractive notation in the numeral consisting of 3 or more numerals
            {
            System.out.println(numeralsToString.indexOf(subtractiveNumerals[i]));


            // this ensures if numeral appears after notation, it gets position 2

            if (numeralsToString.indexOf(subtractiveNumerals[i])>=0 &&
numeralsToString.indexOf(subtractiveNumerals[i]) + 2<=posValues.length-1)
            {
```

```
                tempA = numeralsToString.indexOf(subtractiveNumerals[i])+2;
                System.out.println("NUMERAL AFTER NOTATION!");
            }

            // at this point we know that subtractive notations are legitimate, hence can perform extra step
            // this ensures if numeral appears before notation, it gets position one place before
            // it should only also go here lets say for example  value before notation is bigger than range of the
subtractive notation
            // for instance MIV it should not go there. Reason is this would ensure for such a numeral as  MIVC it
compares forward as oppose backware
            // unsure if this logic will hold.. perhaps better to do

            //if (numeralsToString.indexOf(subtractiveNumerals[i])!=0
            if (numeralsToString.indexOf(subtractiveNumerals[i])!=0 /*&&
posValues[temp][1]>subtractiveNumeralsRange[i][1]  */  )
            {
                tempB = numeralsToString.indexOf(subtractiveNumerals[i])-1;
                valueBeforeSubtractiveNotation = true;
                tempBefore= numeralsToString.indexOf(subtractiveNumerals[i]);
                System.out.println("NUMERAL BEFORE NOTATION!");
            }

            //temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 2;  // sets temp to allow comparison 1st
with 3rd char
            System.out.println("vcbvbnm,,");
            System.out.println(posValues.length-1);
            System.out.println(numeralsToString.indexOf(subtractiveNumerals[i]));


            // NEED MORE LOGIC HERE TO CHANGE HOW TEMP IS GENERATED
            // FOR INSTANCE WITH XCM  temp is one places before than location of index notation


             //HOWEVER WITH  MCM   the M is one place before subtractive notation
             //So need to check in both directions

             System.out.println("first");
            }

            // this is length of two inputtedNumerals. It has no bearing on above.
            if (numeralsToString.indexOf(subtractiveNumerals[i])!=-1 && posValues.length==2) //this should
allow normal flow of app?
            {
                temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
                System.out.println("second");
            }

             System.out.println("temp is currrentl: " + temp);
             System.out.println("WWWWWWWWWWWWWWWWWWWWWWWWW");
             System.out.println("What is I:" + i);

             if (temp<=posValues.length && posValues.length>2 )
             // this ensures that there is a value in numeral to compare against
             // it is set to minumum 3 numerals

            {  //This checks   XCX    first char of subtractivenotation X   against  2 positions in main array

                //System.out.println("B:" + numeralsToString.charAt(temp-1));

                //now need to know the range of the subtractive notation and also the non notation
                // which is either before or after subtractive notation
                // VERY IMPORTANT

                //since the code does not include array holding the numeral and its decimal value, switch
statement
```

```
                // will need to be used. posValues was declared two dimensional holding.
                //Also since something like XC, the index will be split in two and lower and upper range will be
outputted.

                //range for subtractive notation

                switch (subtractiveNumerals[i].charAt(0))
                {
                   case 'I':
                        rangeLower=1;
                        break;

                  case 'V':
                     rangeLower=5;
                        break;

                  case 'X':
                     rangeLower=10;
                        break;

                  case 'L':
                     rangeLower=50;
                        break;

                  case 'C':
                     rangeLower=100;
                        break;

                  case 'D':
                     rangeLower=500;
                        break;

                  case 'M':
                     rangeLower=1000;
                        break;

                  default:
                  System.out.println("Do not reach here");

                }

                switch (subtractiveNumerals[i].charAt(1))
                {
                   case 'I':
                        rangeUpper=1;
                        break;

                  case 'V':
                     rangeUpper=5;
                        break;

                  case 'X':
                     rangeUpper=10;
                        break;

                  case 'L':
                     rangeUpper=50;
                        break;

                  case 'C':
                     rangeUpper=100;
                        break;

                  case 'D':
                     rangeUpper=500;
```

```
                    break;

                case 'M':
                  rangeUpper=1000;
                    break;

                default:
                System.out.println("Do not reach here");

               }


                // This is now the example for XC   X   X=10   range for XC (is  10-90 )
                // So evaluating that value at location is not greater or equal to 10

                //if (tempBefore>0)  // this is showing there is check backwards
                //examples would be CCM

            if (/*subtractiveNumerals[i].charAt(0) == numeralsToString.charAt(tempBefore)  &&*/
posValues[tempB][1] <(subtractiveNumeralsRange[i][1]) && tempBefore>tempB)
            {

                System.out.println("2. A numeral (a letter) of lower value" + numeralsToString.charAt(tempB) +
"(=" + posValues[tempB][1] + ") "+  "cannot precede a group of numerals written in subtractive notation,");
                System.out.println(subtractiveNumerals[i] + "( ="+ subtractiveNumeralsRange[i][1] +").");

                //C   CM: A numeral (a letter) of lower value, C ( = 100), cannot precede a group of
                //numerals written in subtractive notation, CM ( = 900).

                illegalSubtractive=true;


            }

                //temp is set +2 from index of start subtractive notation
                 if (posValues[tempA][1] >=subtractiveNumeralsRange[i][0] && posValues[tempA][1]
<=subtractiveNumeralsRange[i][1] && tempA>tempBefore)  /*&& posValues.length>2*/
            {
                System.out.println("%*%*%*");

                System.out.println("1. The numeral " + numeralsToString.charAt(tempA) + "(=" +
posValues[tempA][1] + ") " + " cannot be placed after a group of numerals written in subtractive notation: " +
subtractiveNumerals[i]);
                System.out.println("Of the same range: " +  "( = " + subtractiveNumeralsRange[i][0] + "-" +
subtractiveNumeralsRange[i][1] + ").");

                //The numeral X ( = 10) cannot be placed after a group of numerals written in subtractive notation,
XC ( = 90),
                //of the same range value (10 - 90)

                illegalSubtractive=true;
            }

            // this is now using scenario such as C CM  where  C (100)  appears before    CM(subtractive numeral) -
range (100-900)
            // C can not be lower than 900(upper range).  It can be equal for scenarios such as MCM (this is valid
1900).

            // This is using IV C as an EXAMPLE

            //1 has been subtracted from posValues since it is zero index based

            //This in the if loop will ensure instances such as M CM are not trapped
            //However C CM is trapped
```

```
            // RangeUpper!= posValues[temp-1][1]

            //using real example IVC that ends up in here

                //this rule has to work in tempB and tempA
                //example is MIVC  since it will pass on MIV section  but fail on IVC section


            // this is looking at perspective of   M   IV  C   (examining IV in relation to C)
             if (subtractiveNumeralsRange[i][1] < posValues[tempA][1] && tempBefore<tempA)  /*&&
posValues.length>2*/
            {
               System.out.println("%*%*%*");

            System.out.println("3ver1. A group of numerals written in subtractive notation, of lower value,");
            System.out.println(subtractiveNumerals[i] + " ( = " + (rangeUpper-rangeLower) + "), cannot precede a
numeral of larger value, "   +  numeralsToString.charAt(tempA) + "( = "+posValues[tempA][1]+").");

            //IV  C: A group of numerals written in subtractive notation, of lower value, IV ( = 4),
            //cannot precede a numeral of larger value, C ( = 100)

               //System.out.println("1. The numeral " + numeralsToString.charAt(temp) + "(=" +
posValues[temp][1] + ") " + " cannot be placed after a group of numerals written in subtractive notation: " +
subtractiveNumerals[i]);
               //System.out.println("Of the same range: " +  "( = " + rangeLower + "-" + rangeUpper + ").");

               illegalSubtractive=true;
            }


            // this is looking at perspective of   M   IV  C   (examining IV in relation to M)
             if (subtractiveNumeralsRange[i][1] > posValues[tempB][1] && tempBefore>tempB)  /*&&
posValues.length>2*/
            {
               System.out.println("%*%*%*");

            System.out.println("3ver2. A group of numerals written in subtractive notation, of higher value,");
            System.out.println(subtractiveNumerals[i] + " ( = " + (rangeUpper-rangeLower) + "), cannot proceed a
numeral of lower value, "   +  numeralsToString.charAt(tempB) + "( = "+posValues[tempB][1]+".");

            //IV  C: A group of numerals written in subtractive notation, of lower value, IV ( = 4),
            //cannot precede a numeral of larger value, C ( = 100)

               illegalSubtractive=true;
            }

            // THIS CODE IS REQUIRED FOR EXAMPLE SUCH AS IVC


            //CAREFUL
           // this is now checking if such a sequence exists CMM. This is not legal since
           //subtractive notation CM (900) is less than M (1000).

               switch (subtractiveNumerals[i])
               {
                  case "IV":
                     rule4Single = rule4Single + 4;
                     break;

                  case "IX":
                     rule4Single = rule4Single + 9;
                     break;

                  case "XL":
                     rule4Single = rule4Single + 40;
```

```
                break;

            case "XC":
                rule4Single = rule4Single + 90;
                break;

            case "CD":
                rule4Single = rule4Single + 400;
                break;

            case "CM":
                rule4Single = rule4Single + 900;
                break;

            default:
            invalidSubtractiveNumeral = true;
          }


          twoSubtractiveNotations=false;
          singleSubtractiveNotations=true;
      }


     processedConversion[numeralsToString.indexOf(subtractiveNumerals[i])]=1;
         temp = numeralsToString.indexOf(subtractiveNumerals[i]) + 1;
         processedConversion[temp]=1;

      }


   }
}


//  MAIN PROGRAM EXECUTION
//this will examine decimal number stored. If identical numerals in consecutive it will add to the total.

for (int m=1; m<posValues.length; m++)  // this is starting not at zero index since compared to previous value
{
 if (processedConversion[m]!=1)
   {
    //posValues[m][0]=numeral position     //[m][1]=decimal
   // This section is best completed examining string of the character array of inputted numerals.

   // Rule 3: The letters V, L, and D are not repeated.
   // There are acceptedNumerals[1], acceptedNumerals[3], acceptedNumerals[5]
   // posValues[m][0] stores information relevant

   //{'M','D','C','L','X', 'V','I'};

   System.out.println("Going into rule 3");

   if (!VLDcheck)
   {
   for (indexCount=0; indexCount<posValues.length; indexCount++)
   {
      System.out.println("value of indexCount: " + indexCount);
      System.out.println(posValues.length);

   if (posValues[indexCount][1]==5)
   {
      countV++;
      System.out.println("found a V");
   }
```

```
        if (posValues[indexCount][1]==50)
        {
           countL++;
        }

        if (posValues[indexCount][1]==500)
        {
           countD++;
        }
        }
        }

        VLDcheck=true;


          //Rule 1: When certain numerals are repeated, the number represented by them is their sum. For
example, II = 1 + 1 = 2, or XX = 10 + 10 = 20, or, XXX = 10 + 10 + 10 = 30.


          if (posValues[m][1]==posValues[m-1][1])   // this is comparing decimal number for numeral against
previous one
          {

           // Rule 2: It is to be noted that no Roman numerals can come together more than 3 times. For example,
we cannot write 40 as XXXX
          // This needs to work from ZeroIndex not m

           System.out.println("starting rule 1");
           //if (m!=(posValues.length-2) && posValues.length>3)   // what does this rule mean, first part

            if (posValues.length>3)


          // m is current position in the numerals array

            {
               // has to be 3 advancing numerals to check if 4 together
            if (m<=posValues.length-3)  // can not check for three in row if last element in array
            {


            //if (temp1+2<=posValues.length)
            //{

               // checking for three in row same

               if (temp1+3<=posValues.length)
               {
                  for (int counter=0;counter<=posValues.length-3;counter=counter+4)
                  {
                     System.out.println("Value of coutner: " + counter);

               if (posValues[temp1][1]==posValues[temp1+1][1] &&
posValues[temp1][1]==posValues[temp1+2][1]  &&  posValues[temp1][1]==posValues[temp1+3][1])
          {
               System.out.println("Four in a row occurrence of: ");
               illegalFourInARowState=true;


               System.out.println("*****************************************************");
            System.out.println("This is temp1 99:" + temp1);
            System.out.println("*****************************************************");

               threeInRow = (posValues[temp1][1] * 3) +threeInRow;
```

```
      temp1 = temp1+4;

   System.out.println("*********************************************************");
   System.out.println("This is new temp1 :" + temp1);
   System.out.println("*********************************************************");

   if (temp1+3>posValues.length)
   {
      break;
   }

   }
   }
   }


}
   }

if (!illegalFourInARowState)
{
   System.out.println("Amit Amlani");

   if (processedConversion[m]!=1 && processedConversion[m-1]!=1)
   {
   rule2total=posValues[m][1]+posValues[m-1][1];
     rule2=rule2+rule2total;    // they will be totalled if same
   System.out.println("So far sum:" + rule2);
   rule2State=true;
   }

   valueConsecutiveOccurences[m] = posValues[m][1];  //storing decimal value in array.

   consecutiveOccurrences=consecutiveOccurrences+2;

   // valueConsecutiveOccurences[m] this keeps a track of the roman numeral that has been repeating
   //num++;

   // this checks if there has already been consecutive occurence of the numeral adjacently
   //[m][0]=numeral position     //[m][1]=decimal

   if (m!=(posValues.length-1))
   {

   if (valueConsecutiveOccurences[m]==posValues[m+1][1] && processedConversion[m+1]!=1)
   {

      System.out.println("does this happen45");
      rule2=rule2-(rule2total/2);
      rule2total=0;


   }
}


// NEED TO VERIFY THIS
  if (illegalThreeInARowState)
{
   rule2=0;
}

}
```

```
        }

        //Rule 5: When a Roman numeral is placed after another Roman numeral of greater value,
        //the result is the sum of the numerals. For example, VIII = 5 + 1 + 1 + 1 = 8, or,
        //XV = 10 + 5 = 15,
        // This appears to be where the code has determined if numerals are in some decent order

        System.out.println("What is value of zeroindexcount: " + zeroIndexCount);

        if (posValues[m][1]<posValues[m-1][1])
        {
           System.out.println("How many times does it enter in this section");

           System.out.println("What is m: " + posValues[m][1]);
           System.out.println("What is m-1: " + posValues[m-1][1]);


           //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])][1]]);

           //********************Need to be careful here of making it already set to 1
           //

           //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])-1][1]]);

           if (processedConversion[m-1]!=1)
           {

           rule5total = posValues[m][1] + posValues[m-1][1];
           rule5 = rule5total + rule5;
           System.out.println("*******RULE5******: " + rule5);
           rule5total = 0;
           rule5State=true;
           //processedConversion[m]=1;

           //UNSURE
           //processedConversion[m-1]=1;


           if (posValues.length==2)
           {
              //processedConversion[m-1]=1;
           }

           if (m!=1)   // if not the first numeral
           {

           if (rule4Single!=0)
           {
              System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!");
              //rule5=rule4Single + posValues[m][1];
              rule5=posValues[m][1] + posValues[m-1][1];
           }



        }

        }
        }
        System.out.println("at rule 6: " + total1);

       //Rule 6: When a Roman numeral is placed before another Roman numeral of greater value,
       //the result is the difference between the numerals. For example, IV = 5 - 1 = 4, or, XL = 50 - 10 = 40,
       //or XC = 100 - 10 = 90
```

```
            System.out.println("Rule 6");
            System.out.println("Must WORK!!!!");

            System.out.println(zeroIndexCount);
            System.out.println(posValues[zeroIndexCount][1]);

            if (posValues[zeroIndexCount][1]<posValues[zeroIndexCount+1][1])
            {
               if (posValues[zeroIndexCount][1]==1 && posValues[zeroIndexCount+1][1]>10)
               {
                  System.out.println("Roman Numeral I can not precede: " + posValues[zeroIndexCount+1][1]);
                  IplacedIncorrect=true;
               }

               if (posValues[zeroIndexCount][1]==10 && posValues[zeroIndexCount+1][1]>100)
               {
                  System.out.println("Roman Numeral X can not precede: " + posValues[zeroIndexCount+1][1]);
                  IplacedIncorrect=true;
               }

               if (posValues[zeroIndexCount][1]==50 && posValues[zeroIndexCount+1][1]>=50)
               {
                  System.out.println("Roman Numeral L can not precede: " + posValues[zeroIndexCount+1][1]);

                  IplacedIncorrect=true;
               }

               if (posValues[zeroIndexCount][1]==5 &&
posValues[zeroIndexCount+1][1]>posValues[zeroIndexCount][1])
               {
                  System.out.println("Roman Numeral V can not precede: " + posValues[zeroIndexCount+1][1]);

                  IplacedIncorrect=true;
               }


               if (posValues[zeroIndexCount][1]!=5 && posValues[zeroIndexCount][1]!=50 &&
posValues[zeroIndexCount][1]!=500 && !IplacedIncorrect)

               {


               processedConversion[zeroIndexCount]=1;

               rule6total = posValues[zeroIndexCount+1][1] - posValues[zeroIndexCount][1];
               rule6 = rule6total + rule6;
               rule6total = 0;
               rule6State=true;
            }

            else
            {  System.out.println("Illegal subtractive notation found with V, D or L");
               rule5=0;
               illegalSubtractive=true;
            }
            }

           // Rule 7: When a Roman numeral of a smaller value is placed between two numerals of greater value,
           //it is subtracted from the numeral on its right. For example, XIV = 10 + (5 - 1) = 14,
           //or, XIX = 10 + (10 - 1) = 19

            System.out.println("Rule 7");
            // rule 7 will not stall by rule 4  since the the right hand side character in rule 4 is always bigger.
```

```
            if (m!=posValues.length && posValues.length>2) // a numeral can not be in middle if it is the last in the
numeral array
            // it also can not be in middle if the first numeral. Hence using m index notation.

            {
               if (processedConversion[m]!=1)
               {

               //START FROM HERE AND TEST  XI

            if (m!=posValues.length-1)
            {


            if (posValues[m][1]<posValues[m-1][1] && posValues[m][1]<posValues[m+1][1])
            {

               //System.out.println("what " + (m-1));

               //processedConversion[m]=1;
               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m])][1]]);
               //processedConversion[m+1]=1;
               //System.out.println("This is value marked with 1: " +
posValues[processedConversion[numeralsToString.indexOf(subtractiveNumerals[m+1])][1]]);

               rule7total = posValues[m+1][1] - posValues[m][1];
               rule7 = rule7total + rule7;
               rule7total = 0;

               System.out.println("total at rule 7:" + rule7);

               }

            }

            }
         }
            zeroIndexCount++;
            temp1++;

            System.out.println("This is overall total:" + (total1));
      }
   }


      // THIS WILL NOW CHECK IF EACH NUMBER HAS OR HAS NOT BEEN PROCESSED IN CHECKING SO far

      int counter=0;
      int unTotal=0;



      for (int c: processedConversion)
      {
         System.out.println("This is value of c: " + c);
         System.out.println("This is the numeral value: " + posValues[counter][1]);

         if (c!=1)
         {
            unTotal = posValues[counter][1] + unTotal;
            System.out.println("This has not been added to the total" + unTotal);
            System.out.println("This is the position: " + counter);
         }
```

```
   counter++;
}



   // this is part of rule 3

   if (countD>1 || countL>1 || countV>1)
   {
      System.out.println("Invalid roman numeral. Numeral V or D or L has occured more than once");
      rule3Fail = true;
   }

    if (inputtedNumerals.length==2)  // checking number with 2 numerals
{
   if (posValues[0][0]==posValues[1][0] && rule3Fail==false)  // if values identical straight forward conversion
   {
      twoNumerals = posValues[0][1] + posValues[1][1];


   System.out.println("Conversion is double:" + (twoNumerals));

   //System.out.println("XL: " +  posValues[0][1]);
   processedConversion[0]=1;

   processedConversion[1]=1;
   //System.out.println("This is value marked with 1: " +  posValues[1][1]);


   }
}


    // THIS WILL NOW CHECK IF EACH NUMBER HAS OR HAS NOT BEEN PROCESSED IN CHECKING SO far


 // it is not picking up MM or II
   // This suggests that they have been marked as processedconversion



    if (!illegalSubtractive && !IplacedIncorrect && !rule3Fail && !illegalFourInARowState)
{
   if (runningTotal>0 && unTotal==0 && FlagSet==true)
   {
      runningTotal = runningTotal / 2;
   }

   validNumber=true;


 System.out.println("\n\n********************* GRAND TOTAL*************");
System.out.println(numeralsToString + " is a VALID roman numeral");
System.out.println("runningTotal:" + runningTotal);
 System.out.println("Not added total:" + unTotal );
System.out.println("**TOTAL: "          + (runningTotal+unTotal));
System.out.println("**********************************");
//System.exit(0);

}

if (!validNumber)
{
 System.out.println("\n\n********************* GRAND TOTAL*************");
```

```
 System.out.println(numeralsToString + " is an INVALID roman numeral");
 System.out.println("Address the issues outputted for a valid Roman numeral");

 System.out.println("**********************************");
 }

    System.out.println("\n\n\n*****************************************");
    System.out.println("OLD PROGRAMME LOGIC");
    System.out.println("\n*****************************************");

 System.out.println(numeralsToString + " is the roman numeral");
 System.out.println("number of Vs: " + countV);
 System.out.println("number of Ls: " + countL);
 System.out.println("number of Ds: " + countD);


 System.out.println("rule4: " + rule4);
 System.out.println("rule2: " + rule2);
 System.out.println("rule4Single: " + rule4Single);
 System.out.println("rule5: " + rule5);
 System.out.println("rule6: " + rule6);
 System.out.println("rule7: " + rule7);
 System.out.println("threeinarow:" + threeInRow);
 System.out.println("Untotal:" + unTotal);

if (rule3Fail)
    {
       System.out.println("More than 2 instances of V, L and D:");
       System.out.println("D: " + countD);
       System.out.println("V: " + countV);
       System.out.println("L: " + countL);
       //System.exit(0);
    }

     if (!illegalSubtractive && !IplacedIncorrect && !illegalThreeInARowState && !rule3Fail)
 {

    if (unTotal!=0 && rule6!=0 && rule5!=0 && rule4Single!=0 && rule2!=0)
     {
        System.out.println("total1_temp: " + (rule4Single + unTotal));
        System.exit(0);
     }


     if (unTotal!=0 && rule6!=0 && rule5!=0 && rule4Single!=0)
     {
        System.out.println("total1: " + (rule4Single + unTotal + rule5));
        System.exit(0);
     }


     if (unTotal!=0 && rule6==0 && rule5==0 && threeInRow!=0)
     {
        System.out.println("total2_temp: " + (rule4Single + unTotal + threeInRow));
        System.exit(0);
     }

    if (rule4Single!=0 && unTotal!=0)
    {
       System.out.println("total2: " + (rule4Single + unTotal));
       System.exit(0);
    }

     if (rule4Single!=0 && unTotal!=0 && rule2!=0 )
    {
```

```
   System.out.println("total4_temp: " + (rule4Single + unTotal));
   System.exit(0);
}


 if (rule4Single==0 && rule2!=0 && rule5!=0 && rule4!=0)
{
   System.out.println("total4: " + (rule4 + rule2 + rule5));
   System.exit(0);
}

 if (rule4Single==0 && rule2!=0 && rule5!=0 && rule4!=0)
{
   System.out.println("total4: " + (rule4 + rule2 + rule5));
   System.exit(0);
}

if (unTotal==0 && rule5==0 && rule2==0 && rule4==0 && threeInRow!=0)
{
   System.out.println("Total5: " + (threeInRow));
System.exit(0);
}

if (unTotal!=0 && rule5!=0 && rule6!=0)
{
System.out.println("Total6: " + (unTotal+rule6));
System.exit(0);
}


  if (unTotal!=0 && rule5!=0 &&rule2!=0)
{
System.out.println("Total6: " + (unTotal));
System.exit(0);
}

 if (unTotal!=0 && rule5==0 && rule2==0 && rule4==0 && threeInRow!=0)
{
   System.out.println("Total7: " + (unTotal+threeInRow));
System.exit(0);
}

if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0 && threeInRow!=0)
{
   System.out.println("Total8: " + (threeInRow+rule2));
   System.exit(0);
}

if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0 && unTotal!=0)
{
   System.out.println("Total9: " + (rule2));
   System.exit(0);
}

if (rule2!=0 && rule4Single==0 && rule5==0 && rule6==0)
{
   System.out.println("Total10: " + rule2);
   System.exit(0);
}

if (rule4Single!=0 && rule5!=0 && rule6!=0)
{
   System.out.println("Total11: " + (rule4Single+rule5));
   System.exit(0);
}
```

```
      if (unTotal!=0 && rule5==0 && rule2==0 && rule4==0 && rule4Single==0)
         {
            System.out.println("Total12: " + (unTotal));
         System.exit(0);
         }

      if (unTotal!=0 && rule4!=0)
         {
         System.out.println("Total13: " + (unTotal + rule4));
         System.exit(0);
         }

       if (rule5!=0 && rule6!=0 && unTotal!=0)
         {
            System.out.println(numeralsToString + "is an invalid roman numeral");
            System.out.println(numeralsToString + "INVESTIGATE CODE OR CONDITIONAL IF LOOPS FOR TOTAL IF
ENTER OCCURS");

            System.exit(0);
         }

       if (rule6!=0 && rule5!=0)
         {
         System.out.println("Total14: " + (rule6+rule5));
         System.exit(0);
         }

      if (rule6!=0 && rule2!=0)
      {
         System.out.println("Total 15: " + (rule6+rule2));
         System.exit(0);
      }

        if (rule5!=0 && rule2!=0 && threeInRow==0 && rule4Single!=0)
      {
         System.out.println("Total16: " + (rule5+rule2));
         System.exit(0);
      }


      if (rule5!=0 && rule2!=0 && threeInRow!=0)
      {
         System.out.println("Total17: " + (rule5+rule2));
         System.exit(0);
      }


       if (rule5!=0 && rule2!=0)
      {
         System.out.println("Total18: " + (rule5+rule2));
         System.exit(0);
      }

      if (rule5!=0 && unTotal!=0)
         {
            System.out.println("Total19: " + (rule5+unTotal));
            System.exit(0);
         }

           if (rule5!=0 && rule4!=0)
         {
            System.out.println("Total20: " + (rule5 + rule4));
            System.exit(0);
         }
```

```
   if (rule5!=0 && threeInRow!=0 && rule4Single!=0)
   {
      System.out.println("Total21: " + (rule5+threeInRow));
      System.exit(0);
   }


   if (rule5!=0 && threeInRow!=0)
   {
      System.out.println("Total22: " + (rule5+threeInRow));
      System.exit(0);
   }

   if (rule5!=0)
   {
      System.out.println("Total23: " + (rule5));
      System.exit(0);
   }

   if (rule2!=0 && unTotal!=0 && rule4Single!=0)
   {
      System.out.println("Total24: " + (rule2 + unTotal + rule4Single));
      System.exit(0);

   }


   if (rule2!=0 && rule4Single!=0)
   {
      System.out.println("Total25: " + (rule2 + rule4Single));
      System.exit(0);
   }

  if (rule2!=0)
   {
      System.out.println("Total26: " + (twoNumerals));
      System.exit(0);
   }


   if (rule4Single!=0)
   {
      System.out.println("Total27 is: " + (unTotal + rule4Single));
      System.exit(0);
   }

     if (rule4!=0)
   {
      System.out.println("Total28 is: " + (rule4));
      System.exit(0);
   }

}

System.out.println(numeralsToString +  " is NOT a VALID roman numeral");
   System.out.println("Address the issues outputted for a valid Roman numeral");
```

```
      //System.out.println("this is the overall sum: " + (rule2+rule4Single+rule5+rule6+rule7));
     // System.out.println("this is the overall sum: " + (rule5 + rule2));

      // This is the best logic found on the internet for roman numeral conversion.
      // Logic will be coded one by one and see if it works overall
   /*

   It is necessary for us to remember the rules for reading and writing Roman numbers in order to avoid mistakes. Here is a list of the basic rules for Roman numerals.

   Rule 1: When certain numerals are repeated, the number represented by them is their sum. For example, II = 1 + 1 = 2, or XX = 10 + 10 = 20, or, XXX = 10 + 10 + 10 = 30.
   Rule 2: It is to be noted that no Roman numerals can come together more than 3 times. For example, we cannot write 40 as XXXX
   Rule 3: The xcletters V, L, and D are not repeated.
   Rule 4: Only I, X, and C can be used as subtractive numerals. There can be 6 combinations when we subtract. These are IV = 5 - 1 = 4; IX = 10 - 1 = 9; XL = 50 - 10 = 40; XC = 100 - 10 = 90; CD = 500 - 100 = 400; and CM = 1000 - 100 = 900
   Rule 5: When a Roman numeral is placed after another Roman numeral of greater value, the result is the sum of the numerals. For example, VIII = 5 + 1 + 1 + 1 = 8, or, XV = 10 + 5 = 15,
   Rule 6: When a Roman numeral is placed before another Roman numeral of greater value, the result is the difference between the numerals. For example, IV = 5 - 1 = 4, or, XL = 50 - 10 = 40, or XC = 100 - 10 = 90
   Rule 7: When a Roman numeral of a smaller value is placed between two numerals of greater value, it is subtracted from the numeral on its right. For example, XIV = 10 + (5 - 1) = 14, or, XIX = 10 + (10 - 1) = 19
   Rule 8: To multiply a number by a factor of 1000 a bar is placed over it.
   Rule 9: Roman numerals do not follow any place value system.
   Rule 10: There is no Roman numeral for zero (0).
*/

   }

}
```