# Please review the following code

The codes intent is to have a class that takes a number and returns the roman numberal string.

You task is to code review the code and identify issues that you would like to change keeping in mind coding best practices.

package converters;  
  
public class RomanNumber  
{  
 private int number;  
  
 private static final int maxValue = 3000;  
 public RomanNumber(int number) throws Exception  
 {  
 if(number > maxValue) throw new Exception("RomanNumber only supports numbers up to 3000");  
 this.number = number;  
 }  
  
 public void SetNumber(String number) throws Exception  
 {  
 this.number = Integer.parseInt(number);  
 if(this.number > maxValue) throw new Exception("RomanNumber only supports numbers up to 3000");  
  
 }  
  
 public String convert()  
 {  
 String result = "";  
  
 try {  
 int thousands = this.number / 1000;  
 result += times(thousands, "M");  
 int hundreds = this.number / 100 % 10;  
 result += times(hundreds, "C", "D", "M");  
 int tens = this.number / 10 % 10;  
 result += times(tens, "X", "L", "C");  
 int ones = this.number % 10;  
 result += times(ones, "I", "V", "X");  
 } catch (Exception ex){  
 System.out.println("An error occured");  
 }  
 }  
  
 private String times(int number, String character)  
 {  
 String result = "";  
  
 for(int i = 0; i < this.number; i++)  
 {  
 result += character;  
 }  
 return result;  
 }  
 private String times(int number, String o, String f, String t) throws Exception  
 {  
 switch(number)  
 {  
 case 0:  
 return "";  
 case 1:  
 case 2:  
 case 3:  
 return times(number, o);  
 case 4:  
 return o + f;  
 case 5:  
 case 6:  
 case 7:  
 case 8:  
 return f + times(number - 5, o);  
 case 9:  
 return o + t;  
 default:  
 throw new Exception("Only single digits allowed - not " + number);  
 }  
 }  
}

**Issues Found :**

1. **nonexpressive variables names.**
2. **Wrong name format for final variables.**
3. **nonexpressive methods names.**
4. **Error in existing logic in extracting thousands numbers**

**Refactored Code**

**public** **class** RomanNumber

{

**private** **int** number;

**private** **static** **final** **int** ***MAX\_VALUE*** = 3000;

**public** RomanNumber(**int** number) **throws** Exception

{

**if**(number > ***MAX\_VALUE***) **throw** **new** Exception("RomanNumber only supports numbers up to 3000");

**this**.number = number;

}

**public** **void** SetNumber(String number) **throws** Exception

{

**this**.number = Integer.*parseInt*(number);

**if**(**this**.number > ***MAX\_VALUE***) **throw** **new** Exception("RomanNumber only supports numbers up to 3000");

}

**public** String convertToRomanNum()

{

String result = "";

**try** {

//convert thousands values

**int** thousands = **this**.number / 1000;

result += accumulateCharacterts(thousands, "M");

//convert hundreds values

**int** hundreds = **this**.number / 100 % 10;

result += accumulateCharacterts(hundreds, "C", "D", "M");

//convert tens values

**int** tens = **this**.number / 10 % 10;

result += accumulateCharacterts(tens, "X", "L", "C");

//convert ones values

**int** ones = **this**.number % 10;

result += accumulateCharacterts(ones, "I", "V", "X");

} **catch** (Exception ex){

System.***out***.println("An error occured");

}

**return** result;

}

**private** String accumulateCharacterts(**int** extractedNumber, String character)

{

//

String result = "";

**for**(**int** index = 0; index < extractedNumber; index++)

{

result += character;

}

**return** result;

}

**private** String accumulateCharacterts(**int** extractedNumber, String firstChar, String secondChar, String thirdChar) **throws** Exception

{

**switch**(extractedNumber)

{

**case** 0:

**return** "";

**case** 1:

**case** 2:

**case** 3:

**return** accumulateCharacterts(extractedNumber, firstChar);

**case** 4:

**return** firstChar + secondChar;

**case** 5:

**case** 6:

**case** 7:

**case** 8:

**return** secondChar + accumulateCharacterts(extractedNumber - 5, firstChar);

**case** 9:

**return** firstChar + thirdChar;

**default**:

**throw** **new** Exception("Only single digits allowed - not " + extractedNumber);

}

}

}